Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра экономической информатики

Библиотека БГУИР

ОСНОВЫ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ

Лабораторный практикум

для студентов специальности I-40 01 02-02
«Информационные системы и технологии в экономике»
всех форм обучения

В 2-х частях

Часть 2

Минск 2007

УДК 002.5 + 681.3.06(075.8)
ББК 32.81 + 32.973.26 – 018 я 73
О-66

Р е ц е н з е н т
зав. кафедрой математики и информатики
Минского филиала Московского государственного
университета экономики, статистики и информатики,
канд. техн. наук, доцент В. П. Васильев

А в т о р ы:
Е. Н. Живицкая, В. Н. Комличенко,
С. Н. Кардаш, Н. А. Кириенко,
И. П. Логинова, Д. А. Сторожев,
Н. П. Мытник, А. А. Тарасевич

О-66 **Основы** информатики и программирования : лаб. практикум для студ. спец. I-40 01 02-02 «Информационные системы и технологии в экономике» всех форм обуч. В 2 ч. Ч. 2 / Е. Н. Живицкая [и др.]. – Минск : БГУИР, 2007. – 108 с.
ISBN 978-985-488-180-5 (ч. 2).

В практикуме представлены основные темы лекционного курса «Основы информатики и программирования», методические рекомендации, примеры программной реализации типового задания, список используемой литературы и задания для лабораторных работ.

УДК 002.5 + 681.3.06(075.8)
ББК 32.81 + 32.973.26 – 018 я 73

ISBN 978-985-488-180-5 (ч. 2)
ISBN 978-985-488-179-9

© УО «Белорусский государственный университет информатики и радиоэлектроники», 2007

Библиотека БГУИР

Содержание

Лабораторная работа №9. Классы хранения и видимость переменных 4
Лабораторная работа №10. Препроцессорные средства................................ 20
Лабораторная работа №11. Динамическое распределение памяти............... 46
Лабораторная работа №12. Структура данных «список»............................... 57
Лабораторная работа №13. Очереди. Операции над очередями. Деки.......... 71
Лабораторная работа №14. Стеки. Очереди. Операции над стеками
и очередями.. 81
Лабораторная работа №15. Алгоритмы методов сортировки и поиска......... 89
Лабораторная работа №16. Программирование алгоритмов
вычислительной математики 103
Литература.. 108

Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №9

КЛАССЫ ХРАНЕНИЯ И ВИДИМОСТЬ ПЕРЕМЕННЫХ

Цель: Ознакомиться с классами хранения, областью видимости и временем жизни программных объектов.

Теоретические сведения

1. Исходные файлы и объявление переменных

Обычная Си-программа представляет собой определение функции main, которая для выполнения необходимых действий вызывает другие функции. Приведенные в части 1 примеры программ представляли собой один исходный файл, содержащий все необходимые для выполнения программы функции. Связь между функциями осуществлялась по данным посредством передачи параметров и возврата значений функций. Но компилятор языка Си позволяет также разбить программу на несколько отдельных частей (исходных файлов), оттранслировать каждую часть отдельно и затем объединить все части в один выполняемый файл при помощи редактора связей. При такой структуре исходной программы функции, находящиеся в разных исходных файлах, могут использовать глобальные внешние переменные. Все функции в языке Си по определению внешние и всегда доступны из любых файлов. Например, если программа состоит из двух исходных файлов, как показано на рис. 9.1, то функция main может вызывать любую из трех функций fun1, fun2, fun3, а каждая из этих функций может вызывать любую другую.

```
main ()                  fun2()
{ ...                    { ...
}                        }
fun1()                   fun3()
{ ...                    { ...
}                        }
         file1.c                  file2.c
```

Рис. 9.1. Пример программы из двух файлов

Для того чтобы определяемая функция могла выполнять какие-либо действия, она должна использовать переменные. В языке Си все переменные должны быть объявлены до их использования. Объявления устанавливают соответствие имени и атрибутов переменной, функции или типа. Определение

Библиотека БГУИР

переменной вызывает выделение памяти для хранения ее значения. Класс выделяемой памяти определяется спецификатором класса памяти и определяет время жизни и область видимости переменной, связанные с понятием блока программы. В языке Си блоком считается последовательность объявлений, определений и операторов, заключенная в фигурные скобки. Существуют два вида блоков – составной оператор и определение функции, состоящее из составного оператора, являющегося телом функции, и предшествующего телу заголовка функции (в который входят имя функции, типы возвращаемого значения и формальных параметров). Блоки могут включать в себя составные операторы, но не определения функций. Внутренний блок называется вложенным, а внешний блок – объемлющим.

Время жизни – это интервал времени выполнения программы, в течение которого программный объект (переменная или функция) существует. Время жизни переменной может быть локальным или глобальным. Переменная с глобальным временем жизни имеет распределенную для нее память и определенное значение на протяжении всего времени выполнения программы, начиная с момента выполнения объявления этой переменной. Переменная с локальным временем жизни имеет распределенную для него память и определенное значение только во время выполнения блока, в котором эта переменная определена или объявлена. При каждом входе в блок для локальной переменной распределяется новая память, которая освобождается при выходе из блока.

Область видимости – это часть текста программы, в которой может быть использован данный объект. Объект считается видимым в блоке или в исходном файле, если в этом блоке или файле известны имя и тип объекта. Объект может быть видимым в пределах блока, исходного файла или во всех исходных файлах, образующих программу. Это зависит от того, на каком уровне объявлен объект: на внутреннем, т.е. внутри некоторого блока, или на внешнем, т.е. вне всех блоков. Если объект объявлен внутри блока, то он видим в этом блоке и во всех внутренних блоках. Если объект объявлен на внешнем уровне, то он видим от точки его объявления до конца данного исходного файла.

Спецификатор класса памяти в объявлении переменной может быть auto, register, static или extern. Если класс памяти не указан, то он определяется по умолчанию из контекста объявления. Объекты классов auto и register имеют локальное время жизни. Спецификаторы static и extern определяют объекты с глобальным временем жизни. При объявлении переменной на внутреннем уровне может быть использован любой из четырех спецификаторов класса памяти, а если он не указан, то подразумевается класс памяти auto. Переменная

Библиотека БГУИР

с классом памяти auto имеет локальное время жизни и видна только в блоке, в котором объявлена. Память для такой переменной выделяется при входе в блок и освобождается при выходе из блока. При повторном входе в блок этой переменной может быть выделен другой участок памяти. Переменная с классом памяти auto автоматически не инициализируется. Она должна быть проинициализирована явно при объявлении путем присвоения ей начального значения. Значение неинициализированной переменной с классом памяти auto считается неопределенным.

Спецификатор класса памяти register предписывает компилятору распределить память для переменной в регистре, если это представляется возможным. Использование регистровой памяти обычно приводит к сокращению времени доступа к переменной. Переменная, объявленная с классом памяти register, имеет ту же область видимости, что и переменная auto. Число регистров, которые можно использовать для значений переменных, ограничено возможностями компьютера, и в том случае, если компилятор не имеет в распоряжении свободных регистров, то переменной выделяется память как для класса auto. Класс памяти register может быть указан только для переменных с типом int или указателей с размером, равным размеру int.

Переменные, объявленные на внутреннем уровне со спецификатором класса памяти static, обеспечивают возможность сохранить значение переменной при выходе из блока и использовать его при повторном входе в блок. Такая переменная имеет глобальное время жизни и область видимости внутри блока, в котором она объявлена. В отличие от переменных с классом auto, память для которых выделяется в стеке, для переменных с классом static память выделяется в сегменте данных, и поэтому их значение сохраняется при выходе из блока.

Пример 9.1

```
/*объявления переменной i на внутреннем уровне с классом памяти static
   исходный файл file1.c   */
main()   {...}
fun1()   {  static int i=0; ...   }
 /*  исходный файл   file2.c  */
fun2()   {  static int i=0; ...   }
fun3()   {  static int i=0; ...   }
```

В приведенном примере объявлены три разные переменные с классом памяти static, имеющие одинаковые имена i. Каждая из этих переменных имеет глобальное время жизни, но видима только в том блоке (функции), в которой

Библиотека БГУИР

она объявлена. Эти переменные можно использовать для подсчета числа обращений к каждой из трех функций.

Переменные класса памяти static могут быть инициализированы константным выражением. Если явной инициализации нет, то такой переменной присваивается нулевое значение. При инициализации константным адресным выражением можно использовать адреса любых внешних объектов, кроме адресов объектов с классом памяти auto, так как адрес последних не является константой и изменяется при каждом входе в блок. Инициализация выполняется один раз при первом входе в блок.

Переменная, объявленная локально с классом памяти extern, является ссылкой на переменную с тем же самым именем, определенную глобально в одном из исходных файлов программы. Цель такого объявления состоит в том, чтобы сделать определение переменной глобального уровня видимым внутри блока.

<u>Пример 9.2</u>

```
/* объявления  переменной  i,  являющейся  именем внешнего массива
длинных целых чисел, на локальном уровне
    исходный файл file1.c                  */
main()   { ...  }
fun1()    { extern long i[]; ...  }
/*   исходный файл file2.c                */
long i[MAX]={0};
fun2()    {  ...   }
fun3()    {  ...   }
```

Объявление переменной i[] как extern в приведенном примере делает ее видимой внутри функции fun1. Определение этой переменной находится в файле file2.c на глобальном уровне и должно быть только одно, в то время как объявлений с классом памяти extern может быть несколько.

Объявление с классом памяти extern требует при необходимости использовать переменную, описанную в текущем исходном файле, но ниже по тексту программы, т.е. до выполнения ее глобального определения. Следующий пример иллюстрирует такое использование переменной с именем st.

<u>Пример 9.3</u>

```
main()   { extern int st[]; ...    }
static int st[MAX]={0};
fun1()    {   ...   }
```

Библиотека БГУИР

Объявление переменной со спецификатором extern информирует компилятор о том, что память для переменной выделять не требуется, так как это выполнено где-то в другом месте программы.

При объявлении переменных на глобальном уровне может быть использован спецификатор класса памяти static или extern, а также можно объявлять переменные без указания класса памяти. Классы памяти auto и register для глобального объявления недопустимы. Объявление переменных на глобальном уровне – это или определение переменных, или ссылки на определения, сделанные в другом месте программы. Объявление глобальной переменной, которое инициализирует эту переменную (явно или неявно), является определением переменной. Определение на глобальном уровне может задаваться в следующих формах:

1. Переменная объявлена с классом памяти static. Такая переменная может быть инициализирована явно константным выражением или по умолчанию нулевым значением. То есть обявления static int i=0 и static int i эквивалентны, и в обоих случаях переменной i будет присвоено значение 0.

2. Переменная объявлена без указания класса памяти, но с явной инициализацией. Такой переменной по умолчанию присваивается класс памяти static. То есть объявления int i=1 и static int i=1 будут эквивалентны.

Переменная, объявленная глобально, видима в пределах остатка исходного файла, в котором она определена. Выше своего описания и в других исходных файлах эта переменная невидима (если только она не объявлена с классом extern). Глобальная переменная может быть определена только один раз в пределах своей области видимости. В другом исходном файле может быть объявлена другая глобальная переменная с таким же именем и с классом памяти static, конфликта при этом не возникает, так как каждая из этих переменных будет видимой только в своем исходном файле. Спецификатор класса памяти extern для глобальных переменных используется, как и для локального объявления, в качестве ссылки на переменную, объявленную в другом месте программы, т.е. для расширения области видимости переменной. При таком объявлении область видимости переменной расширяется до конца исходного файла, в котором сделано объявление. В объявлениях с классом памяти extern не допускается инициализация, так как эти объявления ссылаются на уже существующие и определенные ранее переменные. Переменная, на которую делается ссылка с помощью спецификатора extern, может быть определена только один раз в одном из исходных файлов программы.

Библиотека БГУИР

2. Объявление функций

Функции всегда определяются глобально. Они могут быть объявлены с классом памяти static или extern. Объявления функций на локальном и глобальном уровнях имеют одинаковый смысл.

Правила определения области видимости для функций отличаются от правил видимости для переменных и состоят в следующем:

1. Функция, объявленная как static, видима в пределах того файла, в котором она определена. Каждая функция может вызвать другую функцию с классом памяти static из своего исходного файла, но не может вызвать функцию, определенную с классом static, в другом исходном файле. Разные функции с классом памяти static, имеющие одинаковые имена, могут быть определены в разных исходных файлах, и это не ведет к конфликту.
2. Функция, объявленная с классом памяти extern, видима в пределах всех исходных файлов программы. Любая функция может вызывать функции с классом памяти extern.
3. Если в объявлении функции отсутствует спецификатор класса памяти, то по умолчанию принимается класс extern. Все объекты с классом памяти extern компилятор помещает в объектном файле в специальную таблицу внешних ссылок, которая используется редактором связей для разрешения внешних ссылок. Часть внешних ссылок порождается компилятором при обращениях к библиотечным функциям Си, поэтому для разрешения этих ссылок редактору связей должны быть доступны соответствующие библиотеки функций.

Приведем пример программы, текст которой размещен в двух файлах (для связи функций программы используется внешняя переменная i).

Пример 9.4

```
File1.c
#include <stdio.h>
extern int i;
void main(void)
{
    void next(void);
    i++;
    printf("В main i=%d\n",i);
    next();
}
int i=3; /*описание переменной*/
```

Библиотека БГУИР

```
void next(void)
{
       void other(void);
       i++;
       printf("B next i=%d\n",i);
       other();
}

File2.c
#include <stdio.h>
extern int i;
void other(void)
{
       i++;
       printf("B other i=%d\n",i);
}
```

Результат

```
B main i=4
B next i=5
B other i=6
```

Варианты индивидуальных заданий

В каждом задании по возможности ввод, вывод, сортировку оформить как отдельные функции (применить к этим функциям классы хранения), функцию сортировки массива структур поместить в отдельный файл, структурную переменную объявить как статическую, счетчики циклов объявить как регистровые переменные, передачу значений осуществить через внешние переменные, осуществить динамическое распределение памяти под структуры.

1. Описать структуру с именем STUDENT, содержащую следующие поля: фамилия и инициалы, номер группы, успеваемость (массив из пяти элементов). Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в массив из десяти структур типа STUDENT; упорядочить записи по алфавиту; вывод на дисплей фамилий и номеров групп для всех студентов, имеющих оценки 8 и 9; при отсутствии таких студентов вывести соответствующее сообщение.

Библиотека БГУИР

2. Описать структуру с именем AEROFLOT, содержащую следующие поля: название пункта назначения рейса, номер рейса, тип самолета. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в массив, состоящий из семи элементов типа AEROFLOT; упорядочить записи по алфавиту; вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры; при отсутствии таких рейсов выдать на дисплей соответствующее сообщение.

3. Описать структуру с именем WORKER, содержащую следующие поля: фамилия и инициалы работника; название занимаемой должности; год поступления на работу. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в массив, состоящий из десяти структур типа WORKER; упорядочить записи по алфавиту; вывод на дисплей фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры; при отсутствии таких работников вывести на дисплей соответствующее сообщение.

4. Описать структуру с именем TRAIN, содержащую следующие поля: название пункта назначения; номер поезда; время отправления. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в массив из шести элементов типа TRAIN; упорядочивать записи по алфавиту; вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры; при отсутствии таких поездов выдать на дисплей соответствующее сообщение.

5. Описать структуру с именем MARSH, содержащую следующие поля: название начального пункта маршрута; название конечного пункта маршрута; номер маршрута. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в массив, состоящий из восьми элементов типа MARSH; упорядочивать записи по алфавиту; вывод на экран информации о маршрутах, которые начинаются или кончаются в пункте, название которого введено с клавиатуры; при отсутствии таких маршрутов выдать на дисплей соответствующее сообщение.

6. Описать структуру с именем NOTE, содержащую следующие поля: фамилия, имя; номер телефона; день рождения (массив из трех чисел). Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в массив, состоящий из восьми элементов типа NOTE; упорядочить записи по датам дней рождения; вывод на экран информации о человеке, номер телефона которого введен с клавиатуры; при отсутствии такого человека выдать на дисплей соответствующее сообщение.

Библиотека БГУИР

7. Описать структуру с именем ZNAK, содержащую следующие поля: фамилия, имя; знак зодиака; день рождения (массив из трех чисел). Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в массив, состоящий из восьми элементов типа ZNAK; упорядочивать записи по датам дней рождения; вывод на экран информации о людях, родившихся под знаком, наименование которого введено с клавиатуры; выдать на дисплей соответствующее сообщение при отсутствии таких людей.

8. Разработать программу учета покупок ювелирного магазина. Данные о покупках хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по стоимости ювелирного украшения.

9. Разработать программу учета жилищного фонда. Данные о жилом фонде хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по номеру жилищного договора.

10. Разработать программу учета стройматериалов. Данные о стройматериалах хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по номеру договора.

11. Разработать программу учета посадок на участке в ботаническом саду. Данные об участках хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по номеру участка.

12. Разработать программу расчета закупки сырья промышленного предприятия. Данные о закупках хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по типу сырья.

13. Разработать программу расчета прибыли от выполняемых работ по ремонту офиса многофилиального концерна. Данные о выполняемых работах хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по сумме выполненных работ.

14. Разработать программу расчета деталей, использованных при изготовлении какого-либо изделия. Данные о деталях хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по стоимости деталей, используемых в данном изделии.

15. Разработать программу расчета закупки сырья промышленного предприятия. Данные о закупках хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по номеру накладной.

Библиотека БГУИР

16. Разработать программу определения затрат рабочего времени на выполнение строительных работ. Данные о строительных работах хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по номеру заказа.

17. Разработать программу определения пробега автомобиля на основе путевых листов. Данные о путевых листах хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по номеру путевого листа.

18. Разработать программу определения величины таможенных сборов на базе контрактов коммерческой фирмы. Данные о таможенных сборах хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по номеру контракта.

19. Разработать программу определения процента выхода годных изделий на основе актов приема ОТК. Данные о тестируемых партиях хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по номеру заказа.

20. Разработать программу оценки экспорта фирмы. Данные об экспортных операциях хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по номеру контракта.

21. Разработать программу оценки роста промышленного предприятия по данным за последние годы. Данные о финансовых отчетах хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по номеру финансового документа.

22. Разработать программу оценки зависимости продаж театральных билетов от времени года. Данные о продажах хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по величине прибыли.

23. Разработать программу определения суммарной продажи проездных билетов за определенный месяц. Данные о продажах хранить в виде массива структур. Итоговая информация должна выводиться на экран в виде таблицы, отсортированной по величине прибыли.

24. Разработать программу вывода упорядоченного по алфавиту списка студентов, предусмотрев ввод исходной информации о четырех студентах: фамилия и инициалы, год рождения, год поступления в БГУИР, оценки за первый семестр по предметам: физика, высшая математика, информатика.

25. Разработать программу вывода упорядоченного по году рождения списка студентов, предусмотрев ввод исходной информации о пяти студентах:

Библиотека БГУИР

фамилия и инициалы, год рождения, год поступления в БГУИР, оценки за первый семестр по предметам: физика, высшая математика, информатика.

26. Разработать программу вывода упорядоченного по году поступления списка студентов-отличников, предусмотрев ввод исходной информации о четырех студентах: фамилия и инициалы, год рождения, год поступления в БГУИР, оценки за первый семестр по предметам: физика, высшая математика, информатика.

27. Разработать программу вывода анкетных данных студентов, сдавших сессию на 4 и 5, предусмотрев ввод исходной информации о четырех студентах: фамилия и инициалы, год рождения, год поступления в БГУИР, оценки за первый семестр по предметам: физика, высшая математика, информатика.

28. Разработать программу вывода списка студентов, фамилии которых начинаются с буквы Б, и их оценки по всем предметам. Предусмотреть ввод исходной информации о четырех студентах: фамилия и инициалы, год рождения, год поступления в БГУИР, оценки за первый семестр по предметам: физика, высшая математика, информатика.

29. Разработать программу вывода анкетных данных отличников. Предусмотреть ввод исходной информации о четырех студентах: фамилия и инициалы, год рождения, год поступления в БГУИР, оценки за первый семестр по предметам: физика, высшая математика, информатика.

30. Разработать программу вывода списка студентов, фамилии которых начинаются с буквы А, и их даты рождения. Предусмотреть ввод исходной информации о четырех студентах: фамилия и инициалы, год рождения, год поступления в БГУИР, оценки за первый семестр по предметам: физика, высшая математика, информатика.

31. Разработать программу вывода анкетных данных студентов, имеющих хотя бы одну оценку «3» в сессию. Предусмотреть ввод исходной информации о четырех студентах: фамилия и инициалы, год рождения, год поступления в БГУИР, оценки за первый семестр по предметам: физика, высшая математика, информатика.

32. Разработать программу вывода списка студентов и их оценки, фамилии студентов начинаются с букв В и Г. Предусмотреть ввод исходной информации о четырех студентах: фамилия и инициалы, год рождения, год поступления в БГУИР, оценки за первый семестр по предметам: физика, высшая математика, информатика.

33. Разработать программу вывода фамилий и дат рождения студентов, не имеющих оценок «3». Предусмотреть ввод исходной информации о четырех студентах: фамилия и инициалы, год рождения, год поступления в БГУИР,

Библиотека БГУИР

оценки за первый семестр по предметам: физика, высшая математика, информатика.

34. В программе вычислить общий средний балл всех студентов и вывести на экран список студентов со средними баллами выше общего среднего балла. Предусмотреть ввод исходной информации о пяти студентах: фамилия и инициалы, год рождения, год поступления в БГУИР, оценки за первый семестр по предметам: физика, высшая математика, информатика.

35. Разработать программу вывода анкетных данных студентов, имеющих оценку 4 по физике и оценку 5 по высшей математике. Предусмотреть ввод исходной информации о пяти студентах: фамилия и инициалы, год рождения, год поступления в БГУИР, оценки за первый семестр по предметам: физика, высшая математика, информатика.

Пример программы

Программа состоит из двух файлов: f1.cpp и f2.cpp. Файл f1.cpp содержит описание главной функции, в которой реализовано меню. В меню происходит вызов основных функций, описание которых находится в файле f2.cpp. Кроме того, в файле f2.cpp объявлена структура.

```
//Файл f1.cpp
#include <iostream>
#include "f2.cpp"
extern int kol_zakazov;
int main()
{      auto char choice;
       do   {cout << "(E)nter information.\n";
              cout << "(R)ead information.\n";
              cout << "Re(A)d opredel. information.\n";
              cout << "(S)ort information.\n";
              cout << "(Q)uit from programm.\n\n";
              cout << "Enter your choice: ";
              cin >> choice;
              switch(choice)
              {      case 'e':
                     case 'E': enter();
                     break;
                     case 'r':
                     case 'R': print();
                     break;
```

Библиотека БГУИР

```
                case 'a':
                case 'A': read();
                break;
                case 's':
                case 'S': sort();
                break;
                case 'q':
                case 'Q': return 0;   }
        }
        while(choice != 'Q');
return 0;
}
//Файл f2.cpp
#include <iostream>
#include <cstdio>
#include <cstring>
struct zakaz
{       int nomer_zakaza;
        char nazvanie_siriy[20];
        int kol_vo;
        int stoimost;
        int itog_stoimost;  };
static zakaz data[20], temp[20];
int kol_zakazov;
extern void enter()
{       register int a;
        cout << "Kak mnogo zakazov budete vvodit': ";
        cin >> kol_zakazov;
        for(a=0; a<kol_zakazov; a++)
        {       cout << "\nEnter nomer zakaza: "; cin >> data[a].nomer_zakaza;
                cout << "Enter nazvanie siria: "; cin >> data[a].nazvanie_siriy;
                cout << "Enter kol-vo(v \"kg\" ili \"stuk\"): "; cin >> data[a].kol_vo;
                cout << "Enter stoimost' za \"kg\" ili \"styky\": "; cin >> data[a].stoimost;
                data[a].itog_stoimost = data[a].kol_vo * data[a].stoimost;   }
        cout << '\n';
}
extern void print()
{       register int a;
        cout << '\n';
```


Библиотека БГУИР

```
	cout << "#   Nazv. siria     Kol-vo   Stoimost' Obc. stoimost'\n";
	cout <<"************************************************\n";
	for(a=0; a<kol_zakazov; a++)
	{	printf("%d %10s %18d %16d %16d", data[a].nomer_zakaza,
	data[a].nazvanie_siriy, data[a].kol_vo, data[a].stoimost, data[a].itog_stoimost);
		cout << '\n';	}
	cout << '\n';
}
extern int read()
{  register int a;
	cout << "Vvedite nomer zakaza chtobi prochitat' inform.\n";
	auto int choice; cin >> choice;
	for(a=0; a<kol_zakazov; a++)
	{	if(choice == data[a].nomer_zakaza)
		{	cout << "#   Nazv. siria     Kol-vo   Stoimost' Obc. stoimost'\n";
			cout << "**********************************************\n";
			printf("%d %10s %18d %16d %16d", data[a].nomer_zakaza,
			data[a].nazvanie_siriy,
			data[a].kol_vo, data[a].stoimost, data[a].itog_stoimost);
			cout << "\n\n";
			return 0;   }
	}
	cout << "Takogo nomera zakaza НЕТ!";
}
extern void sort_nomer()
{	register int a, b;
	cout << "СТАРАЯ";
	print();
	for(a=1; a<kol_zakazov; a++) for(b=kol_zakazov-1; b>=a; b--)
	{	if(data[b-1].nomer_zakaza > data[b].nomer_zakaza)
		{	temp[b] = data[b-1];
			data[b-1] = data[b];
			data[b] = temp[b];  }
	}
	cout << "НОВАЯ";
	print();
}
extern void sort_nazvanie()
{  register int a, b;
```

Библиотека БГУИР

```
	cout << "СТАРАЯ";
	print();
	for(a=1; a<kol_zakazov; a++) for(b=kol_zakazov-1; b>=a; b--)
	{	if(strcmp(data[b-1].nazvanie_siriy, data[b].nazvanie_siriy) > 0)
		{	temp[b] = data[b-1];
			data[b-1] = data[b];
			data[b] = temp[b];	}
	}
	cout << "НОВАЯ";
	print();
}
extern void sort_kol()
{  register int a, b;
	cout << "СТАРАЯ";
	print();
	for(a=1; a<kol_zakazov; a++) for(b=kol_zakazov-1; b>=a; b--)
	{	if(data[b-1].kol_vo > data[b].kol_vo)
		{	temp[b] = data[b-1];
			data[b-1] = data[b];
			data[b] = temp[b];	}
	}
	cout << "НОВАЯ";
	print();
}
extern void sort_stoimost()
{  register int a, b;
	cout << "СТАРАЯ";
	print();
	for(a=1; a<kol_zakazov; a++) for(b=kol_zakazov-1; b>=a; b--)
	{	if(data[b-1].stoimost > data[b].stoimost)
		{	temp[b] = data[b-1];
			data[b-1] = data[b];
			data[b] = temp[b];	}
	}
	cout << "НОВАЯ";
	print();
}
extern void sort_itog_stoimost()
{  register int a, b;
```

Библиотека БГУИР

```
	cout << "СТАРАЯ";
	print();
	for(a=1; a<kol_zakazov; a++) for(b=kol_zakazov-1; b>=a; b--)
	{  if(data[b-1].itog_stoimost > data[b].itog_stoimost)
		{	temp[b] = data[b-1];
			data[b-1] = data[b];
			data[b] = temp[b];	}
	}
	cout << "НОВАЯ";
	print();
}
extern int sort()
{  if(!data[0].nomer_zakaza)
	{	cout << "You must enter information.\n\n";
		return 0;	}
	auto int choice;
	do  {cout << "\t1. Sort po \"nomer zakaza\".\n";
		cout << "\t2. Sort po \"nazvanie siria\".\n";
		cout << "\t3. Sort po \"kol-vy\".\n";
		cout << "\t4. Sort po \"stoimosti\".\n";
		cout << "\t5. Sort po \"itogovoi stoimosti\".\n";
		cin >> choice;
		switch(choice)
		{	case 1: sort_nomer();
			break;
			case 2: sort_nazvanie();
			break;
			case 3: sort_kol();
			break;
			case 4: sort_stoimost();
			break;
			case 5: sort_itog_stoimost();
			break;	}
	}
	while(choice > 6);
	cout << "***Complited***\n";
	cout << '\n';
	return 0;
}
```


Библиотека БГУИР

ЛАБОРАТОРНАЯ РАБОТА №10

ПРЕПРОЦЕССОРНЫЕ СРЕДСТВА

Цель: Освоить работу с директивами препроцессора включения файлов, макроподстановки, условной компиляции, многофайловыми проектами и другими возможностями препроцессора.

Теоретические сведения

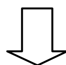
1. Состав директив препроцессора и стадии препроцессорной обработки

В интегрированную среду подготовки программ на Си в качестве обязательного компонента входит препроцессор. Основное назначение препроцессора – анализ и обработка исходного текста программы до ее компиляции. Возможности препроцессора весьма интересны, особенно в областях макрообработки и условной компиляции. Обработка текстов программ – это основная задача препроцессора, однако он может преобразовывать и произвольные тексты.

Итак, на входе препроцессора – текст с препроцессорными директивами, на выходе препроцессора – текст без препроцессорных директив (рис. 10.1).

Текст до препроцессора (исходный текст программы)

```
#define  PI 3.141593
#define ZERO 0.0
if (r > ZERO)  /* Сравнение с константой ZERO */
/* Длина окружности радиуса r:*/
D=2*PI*r;
```



Текст после препроцессора

```
if (r > 0.0) /* Сравнение с константой ZERO */
/*Длина окружности радиуса r: */
D=2*3.141593*r;
```

Рис. 10.1. Обработка текста программы препроцессором

2. Стадии препроцессорной обработки

Препроцессорная обработка текста включает несколько стадий, выполняемых последовательно в следующем порядке:

• все системно зависимые обозначения (например системно зависимый индикатор конца строки) перекодируются в стандартные коды;

Библиотека БГУИР

• каждая пара из символов ‘\’ и “конец строки” вместе с пробелами между ними убирается, и тем самым следующая строка исходного текста присоединяется к строке, в которой находилась эта пара символов;

• в тексте каждой отдельной строки распознаются директивы и лексемы препроцессора, а каждый комментарий заменяется одним символом пустого промежутка;

• выполняются директивы препроцессора и производятся макроподстановки;

• эскейп-последовательности в символьных константах и символьных строках, например ‘\n’ или ‘\xF2’, заменяются на их эквиваленты (на соответствующие числовые коды);

• смежные символьные строки (строковые константы) конкатенируются, т.е. соединяются в одну строку;

• каждая препроцессорная лексема преобразуется в текст на языке Си.

К препроцессорным лексемам или лексемам препроцессора (preprocessing token) относятся: символьные константы, имена включаемых файлов, идентификаторы, знаки операций, препроцессорные числа, знаки препинания, строковые константы (строки) и любые символы, отличные от пробела.

Рассмотрим подробно стадию обработки директив препроцессора. При ее выполнении возможны следующие действия:

• замена идентификаторов (обозначений) заранее подготовленными последовательностями символов;

• включение в программу текстов из указанных файлов;

• исключение из программы отдельных частей ее текста (условная компиляция);

• макроподстановка, т.е. замена обозначения параметризованным текстом, формируемым препроцессором с учетом конкретных параметров.

Перед символом ‘#’ и после него в директиве разрешены пробелы. Пробелы также разрешены перед лексемами препроцессора, между ними и после них. Окончанием препроцессорной директивы служит конец текстовой строки (при наличии символа ‘\’, обозначающего перенос строки, окончанием препроцессорной директивы будет признак конца следующей строки текста).

Определены следующие препроцессорные директивы:

#define – определение макроса или препроцессорного идентификатора;
#include – включение текста из файла;
#undef – отмена определения макроса или идентификатора (препроцессорного);
#if – проверка условия-выражения;
#ifdef – проверка определенности идентификатора;
#ifndef – проверка неопределенности идентификатора;
#else – начало альтернативной ветви для **#if**;
#endif – окончание условной директивы **#if**;

Библиотека БГУИР

#elif – составная директива **#else/#if**;
#line – смена номера следующей ниже строки;
#error – формирование текста сообщения об ошибке трансляции;
#pragma – действия, предусмотренные реализацией;
– пустая директива.

Кроме препроцессорных директив имеются три препроцессорные операции, которые будут подробно рассмотрены вместе с командой **#define**:

defined – проверка истинности операнда;
– конкатенация препроцессорных лексем;
– преобразование операнда в строку символов.

Директива **#define** имеет несколько модификаций. Они предусматривают определение макросов и препроцессорных идентификаторов, каждому из которых ставится в соответствие некоторая символьная последовательность. В последующем тексте программы препроцессорные идентификаторы заменяются заранее запланированными последовательностями символов.

Директива **#include** позволяет включать в текст программы текст из указанного файла.

Директива **#undef** отменяет действие директивы **#define**, которая определила до этого имя препроцессорного идентификатора.

Директива **#if** и ее модификации **#ifdef**, **#ifndef** совместно с директивами **#else**, **#endif**, **#elif** позволяют организовать условную обработку текста программы. При использовании этих средств компилируется не весь текст, а только те его части, которые выделены с помощью перечисленных директив.

Директива **#line** позволяет управлять нумерацией строк в файле с программой. Имя файла и желаемый начальный номер строки указываются непосредственно в директиве **#line**.

Директива **#error** позволяет задать текст диагностического сообщения, которое выводится при возникновении ошибок.

Директива **#pragma** вызывает действия, зависящие от реализации, т.е. запланированные авторами компилятора.

Директива # ничего не вызывает, так как является пустой директивой, т.е. не дает никакого эффекта и всегда игнорируется.

Рассмотрим возможности перечисленных директив и препроцессорных операций.

3. Замены в тексте программы

Директива **#define**. Используется для замены выбранного программистом идентификатора заранее подготовленной последовательностью:

Формат:
#define идентификатор строка замещения

Библиотека БГУИР

Директива может размещаться в любом месте обрабатываемого текста, а ее действие в обычном случае распространяется от позиции размещения директивы до конца файла. В результате работы препроцессора вхождения идентификатора, определенного командой **#define**, в тексте программы заменяются строкой замещения, окончанием которой обычно служит признак конца той «физической» строки, где размещена команда **#define**. Символы пробелов, помещенные в начале и в конце строки замещения, в подстановке не используются.

<u>Пример 10.1</u>

Исходный текст

```
#define begin {
#define end }
void main()
begin
операторы
end
```

Результат препроцессорной обработки

```
void main( )
{
  операторы
}
```

В данном случае используются в качестве операторных скобок идентификаторы **begin**, **end**. Соответствующие указания программист дал препроцессору с помощью директив **#define**. До компиляции препроцессор заменяет все вхождения этих идентификаторов стандартными скобками ‘**{**‘ и ‘**}**’.

4. Цепочка подстановок

Если в строке замещения команды **#define** в качестве отдельной лексемы встречается препроцессорный идентификатор, ранее определенный другой директивой **#define**, то выполняется цепочка последовательных подстановок. В качестве примера рассмотрим, как можно определить диапазон (**RANGE**) возможных значений любой целой переменной типа **int** в следующей программе:

<u>Пример 10.2</u>

```
#include <limits.h>

#define  RANGE   ((INT_MAX) –  < INT_MIN)+1)

/* RANGE  - диапазон значений для int */   ………….

int RANGE_T  =  RANGE/8;   ………….
```

Препроцессор последовательно просматривает текст и, обнаружив директиву **#include <limits.h>**, вставляет текст из файла **limits.h**. Там определены константы INT_MAX и INT_MIN (предельные максимальное и минимальное значения целых величин). Программа принимает следующий вид:

Библиотека БГУИР

```
#define INT_MAX  32767
#define INT_MIN  -32768
#define RANGE   ((INT_MAX)-(INT_MIN)+1).
/* RANGE - диапазон значений для int*/
int RANGE_T = RANGE/8;
```

Обратим внимание на то, что директива **#include** исчезла из программы, но ее заменил соответствующий текст. Обнаружив в тексте (добытом из файла **limits.h**) директивы **define**..., препроцессор выполняет соответствующие подстановки, и программа принимает следующий вид:

```
#define RANGE ((32767)-(-32768)+1)
/* RANGE - диапазон значений для int*/
int RANGE_T = RANGE/8;
```

Подстановки изменили строку замещения препроцессорного идентификатора RANGE в директиве **#define**, размещенной ниже, чем текст, включенный из файла **limits.h**. Последовательно анализируя текст программы, препроцессор встречает препроцессорный идентификатор RANGE и выполняет подстановку. Программа приобретает следующий вид:

```
/* RANGE  - диапазон значений для int*/
int RANGE_T  -   ((32767) - (-32768)+1)/8;
```

Теперь все директивы **#define** удалены из текста. Получен текст, пригодный для компиляции, т.е. создана «единица трансляции». Подстановка строки замещения вместо идентификатора RANGE выполнена в выражении RANGE/8, однако внутри комментария идентификатор RANGE остался без изменений и не изменился идентификатор RANGE_T.

Этот пример иллюстрирует выполнение «цепочки» подстановок и ограничения на замены: замены не выполняются внутри комментариев, внутри строковых констант, внутри символьных констант и внутри идентификаторов (не может измениться часть идентификатора). Например, RANGE_T остался без изменений.

Если строка замещения оказывается слишком длинной, то ее можно продолжить в следующей строке текста. Для этого в конце продолжаемой строки помещается символ **‘\’**. В ходе одной из стадий препроцессорной обработки этот символ вместе с последующим символом конца строки будет удален из текста программы.

Пример 10.3

```
#define STRING "\n Game Over!   –  Игра закончена!"
printf(STRING);
```


Библиотека БГУИР

На экран будет выведено:
Game Over! – Игра закончена!

С помощью команды **#define** удобно выполнять настройку программы. Например, если в программе требуется работать с массивами, то их размеры можно явно определять на этапе препроцессорной обработки:

Пример 10.4

Исходный текст	Результат препроцессорной обработки
#define К 40	
void main( )	void main ( )
{	{
int M[K][K] ;	int M[40][40] ;
float A[2*K+1],	float A[2*40+l] ,
float B[K+3][K-3]	float B[40+3][40-3];
……..	……..

При таком описании очень легко изменять предельные размеры сразу всех массивов, изменив только одну константу (строку замещения) в директиве **#define**. Предусмотренные директивой **#define** препроцессорные замены не выполняются внутри строк, символьных констант и комментариев, т.е. не распространяются на тексты, ограниченные кавычками (“), апострофами (‘) и разделителями (/*,. */). В то же время строка замещения может содержать перечисленные ограничители, например при замене препроцессорного идентификатора STRING.

Если в программе нужно часто печатать или выводить на экран дисплея значение какой-либо переменной и, кроме того, снабжать эту печать одним и тем же пояснительным текстом, то удобно ввести сокращенное обозначение оператора печати, например:

Пример 10.5

```
#define PK printf(“\n Номер элемента=%d.”,N) ;
```

После этой директивы использование в программе оператора РК будет эквивалентно (по результату) оператору из строки замещения. Например, последовательность операторов

```
int N = 4;   PK;
```

приведет к выводу такого текста:

Номер элемента=4.

Если в строку замещения входит идентификатор, определенный в другой команде **#define**, то в строке замещения выполняется следующая замена (цепочка подстановок). Например, программа, содержащая команды:

Библиотека БГУИР

Пример 10.6

```
#define K 50
#define PE printf ("\n Число элементов K=%d",K);
…….
PE;
```

выведет на экран такой текст:

Число элементов K=50

Идентификатор K внутри строки замещения, обрамленной кавычками, не заменяется на 50.

Строку замещения, связанную с конкретным препроцессорным идентификатором, можно сменить, приписав уже определенному идентификатору новое значение другой командой **#define**:

Пример 10.7

```
#define M 16 /*  Идентификатор M определен как 16  */
#define M  'C'  /* M определен как символьная константа   'C' */
#define M "C"  /* M определен как символьная строка, в которой есть два элемента 'C' и '\0' */
```

Однако при таких сменах значений препроцессорного идентификатора некоторые компиляторы С++ выдают предупреждающее сообщение на каждую следующую директиву **#define:**

Warning ..: Redefinition of 'M' is not identical

Замены в тексте можно отменять с помощью команды

#undef идентификатор

После выполнения такой директивы этот препроцессорный идентификатор становится неопределенным, и его можно определять повторно. Например, не вызовут предупреждающих сообщений директивы:

```
#define M 16
#undef M
#define M 'C'
#undef M
#define M "C"
```

Директиву **#undef** удобно использовать при разработке больших программ, когда они собираются из отдельных «кусков текста», написанных в разное время или разными программистами. В этом случае могут встретиться одинаковые обозначения разных объектов. Чтобы не изменять исходных

Библиотека БГУИР

файлов, включаемый текст можно «обрамлять» подходящими директивами **#define**, **#undef** и тем самым устранять возможные ошибки.

Пример 10.8

```
…….
A = 10;    / Основной текст*/
…….
#define A  X
…….
A = 5;    / Включенный текст */
………..
#undef A
………..
B = A;    / Основной текст */
……….
```

При выполнении программы переменная B примет значение 10, несмотря на наличие оператора присваивания A = 5 во включенном тексте.

5. Включение текстов из файлов

Для включения текста из файла используется команда **#include**, имеющая три формы записи:

```
#include < имя_файла > /* Имя в угловых скобках */
#include “имя_файла”   /* Имя в кавычках */
#include имя_макроса  /* Макрос, расширяемый до обозначения файла*/
```

где имя_макроса – это введенный директивой **#define** препроцессорный идентификатор, либо макрос, при замене которого после конечного числа подстановок будет получена последовательность символов <имя_файла> либо “имя_файла”.

Существует правило, что если имя_файла – в угловых скобках, то препроцессор разыскивает файл в стандартных системных каталогах. Если имя_файла заключено в кавычки, то вначале препроцессор просматривает текущий каталог пользователя и только затем обращается к просмотру стандартных системных каталогов.

При необходимости использования в программах средств ввода–вывода в начале текста программы помещают директиву:

#include <stdio.h>

Выполняя эту директиву, препроцессор включает в программу средства связи с библиотекой ввода–вывода. Поиск файла **stdio.h** ведется в стандартных системных каталогах.

Библиотека БГУИР

По принятому соглашению суффикс **.h** приписывается тем файлам, которые содержат прототипы библиотечных функций, а также определения и описания типов и констант, используемых при работе с библиотеками компилятора. Эти файлы принято называть заголовочными.

Кроме такого **stdio.h** в заголовок программы могут быть включены любые другие файлы (стандартные или подготовленные специально). Перечень обозначений заголовочных файлов для работы с библиотеками компилятора утвержден стандартом языка. Ниже приведены названия этих файлов, а также краткие сведения о тех описаниях и определениях, которые в них включены. Большинство описаний – прототипы стандартных функций, в **stdio.h**, определены в основном константы (например **NULL**, **EOF**), необходимые для работы с библиотечными функциями. Все имена, определенные в стандартных заголовочных файлах, являются зарезервированными именами:

assert.h – Диагностика программ
ctype.h – Преобразование и проверка символов
errno.h – Проверка ошибок
float.h – Работа с вещественными данными
limits.h – Предельные значения целочисленных данных
locate.h – Поддержка национальной среды
math.h – Математические вычисления
setjump.h – Возможности нелокальных переходов
signal.h – Обработка исключительных ситуаций
stdarg.h – Поддержка переменного числа параметров
stddef.h – Дополнительные определения
stdio.h – Средства ввода–вывода
stdlib.h – Функции общего назначения (работа с памятью)
string.h – Работа со строками символов
time.h – Определение дат и времени

В конкретных реализациях количество и наименования заголовочных файлов могут быть и другими. Действие включаемого заголовочного файла распространяется на текст программы только в пределах одного модуля от места размещения директивы **#include** и до конца текстового файла (и всех включаемых в программу текстов).

Заголовочные нестандартные файлы оказываются весьма эффективным средством при модульной разработке крупных программ, когда связь между модулями, размещаемыми в разных файлах, реализуется не только с помощью параметров, но и через внешние объекты, глобальные для нескольких или всех модулей. Описания таких внешних объектов (переменных, массивов, структур и т.п.) и прототипы функций помещаются в одном файле, который с помощью директив **#include** включается во все модули, где необходимы внешние объекты. В тот же файл можно включить и директиву подключения файла с описаниями библиотеки функций ввода–вывода. Заголовочный файл может быть, например, таким:

Библиотека БГУИР

Пример 10.9

```
#include <stdio.h> /* Включение средств обмена */
/*  Целые  внешние  переменные   */
extern int ii, jj, 11;
/* Вещественные внешние переменные */
extern float aa, bb;
```

Если в программе используется несколько функций, то часто удобно текст каждой из них хранить в отдельном файле. При подготовке программы в виде одного модуля в нее включаются тексты всех функций с помощью команд **#include**.

Например, запрограммирована задача формирования гистограммы с целью оценки последовательности псевдослучайных чисел. При разработке программы выделены шесть функций и введены внешние переменные, которые должны быть доступны для этих функций. Список неглавных функций (их прототипы) получается таким:

Пример 10.10

```
void count(void);
void estimate (double *, double *, double *);
void compare (double, double);
double gauss (double, double);
int pseudorand (void);
```

В тексты всех перечисленных функций, а также в функцию **main()** помещены прототипы вызываемых функций. Кроме того, в каждой функции описаны со спецификатором **extern** все внешние объекты, доступ к которым нужен в теле функции.

Такое размещение описаний и определений допускает произвольный порядок размещения текстов функций в программном файле. Если в качестве имен файлов с текстами функций программы выбраны идентификаторы, близкие к именам функций, то возможен следующий текст программы (одни процессорные команды):

Пример 10.11

```
#include  “main.c”
#include   “count.c”
#include  “estimate.c”
#include  “compare.c”
#include   “gauss.c”
#include  “pseudo.c”
```

Библиотека БГУИР

Имена файлов соответствуют именам функций с добавкой расширения “**.c**”.

Как иллюстрирует этот пример, исходный текст программы может быть настолько сокращен, что будет содержать только директивы препроцессора.

Препроцессор включает настоящие (реальные) тексты всех функций в программу и как единое целое (модуль) передает полученный текст на компиляцию.

6. Условная компиляция

Директивы ветвлений

Условная компиляция обеспечивается в языке Си следующим набором директив, которые управляют не компиляцией, а препроцессорной обработкой текста:

Формат:
#if целочисленное_константное_выражение
#ifdef идентификатор
#ifndef идентификатор
#else
#endif
#elif

Первые три директивы выполняют проверку условий, две следующие – позволяют определить диапазон действия проверяемого условия. (Директиву **#elif** рассмотрим несколько позже.) Общая структура применения директив условной компиляции такова:

#if… текст_1
#else текст_2
#endif

Конструкция **#else** текст_2 необязательна. Текст_1 включается в компилируемый текст только при истинности проверяемого условия (обозначено многоточием после **#if**). Если условие ложно, то при наличии директивы **#else** на компиляцию передается текст_2. Если директива **#else** и текст_2 отсутствуют, то весь текст от **#if** до **#endif** при ложном условии опускается. Различие между формами команд **#if** состоит в следующем.

В первой из перечисленных директив

#if целочисленное константное выражение

проверяется значение константного выражения, в которое могут входить целые константы и идентификаторы. Идентификаторы могут быть определены на препроцессорном уровне, и тогда их значение определяется подстановками. В противном случае считается, что идентификаторы имеют нулевые значения.

Библиотека БГУИР

Если константное выражение отлично от нуля, то считается, что проверяемое условие истинно.

Пример 10.12
В результате выполнения директив:
5+4
текст_1
#endif
текст_1 всегда будет включен в компилируемую программу.

В директиве
#ifdef идентификатор
проверяется, определен ли с помощью директивы **#define** к текущему моменту идентификатор, помещенный после **#ifdef**. Если идентификатор определен, т.е. является препроцессорным, то текст_1 используется компилятором.

В директиве
#ifndef идентификатор
проверяется обратное условие – истинным считается неопределенность идентификатора, т.е. тот случай, когда идентификатор не был использован в команде **#define** или его определение было отменено командой **#undef**.

Условную компиляцию удобно применять при отладке программ для включения или исключения средств вывода контрольных сообщений.

Пример 10.13
```
#define DEBUG
#ifdef DEBUG
printf (“ Отладочная печать “);
#endif
```

Таких вызовов функции **printf()**, появляющихся в программе в зависимости от определенности идентификатора DEBUG, может быть несколько, и, убрав либо поместив в скобки комментария /*...*/ директиву **#define** DEBUG, сразу же отключаем все отладочные средства.

Файлы, предназначенные для препроцессорного включения в программу, обычно снабжают защитой от повторного включения. Такое повторное включение может произойти, если несколько файлов, в каждом из которых в свою очередь запланировано препроцессорное включение одного и того же файла, объединяются в общий текст программы. Например, такими средствами защиты снабжены все заголовочные файлы стандартной библиотеки.

Библиотека БГУИР

Пример 10.14

Схема защиты от повторного включения может быть такой:

```
/* Файл с именем filename  */
/* Проверка определенности _FILE_NAME  */
#ifndef     _FILE_NAME .../*  Включаемый текст файла filename  */
/* Определение    _FILE_NAME  */
#define  _FILE_NAME
#endif
```

Здесь _FILE_NAME – зарезервированный для файла filename препроцессорный идентификатор, который желательно не использовать в других текстах программы.

Для организации мультиветвлений (аналог операции **switch**) во время обработки препроцессором исходного текста программы введена директива

#elif целочисленное константное выражение

Требования к целочисленному константному выражению те же, что и в директиве **#if**. Структура исходного текста с применением этой директивы такова:

Пример 10.15

#if условие
текст для if
#elif выражение_1
текст_1
#elif выражение _2
текст_2
#else
текст для случая els
#endif

Препроцессор проверяет вначале условие в директиве **#if**. Если оно ложно (равно 0), – вычисляется выражение_1; если при этом оказывается, что и значением выражения_1 также является 0, – вычисляется выражение_2, и т.д. Если все выражения ложны (равны 0), то в компилируемый текст включается текст_для_случая_else. В противном случае, т.е. при появлении хотя бы одного истинного выражения (в **#if** или в **#elif**), начинает обрабатываться текст, расположенный непосредственно за этой директивой, а все остальные директивы не рассматриваются.

Таким образом, препроцессор обрабатывает всегда только один из участков текста, выделенных директивами условной компиляции.

Библиотека БГУИР

7. Операция defined

При условной обработке текста (при условной компиляции с использованием директив **#if**, **#elif)** для упрощения записи сложного условия выбора можно использовать унарную препроцессорную операцию

#defined операнд

где операнд – либо идентификатор, либо заключенный в скобки идентификатор, либо обращение к макросу. Если идентификатор операнда до этого определен с помощью команды **#define** как препроцессорный, то выражение **defined** операнд принимает значение 1L, т.е. считается истинным. В противном случае его значение равно 0L.

Выражение
#if defined операнд
эквивалентно выражению
#ifdef операнд

Но в таком простом случае никакие достоинства операции **defined** не проявляются. Поясним с помощью примера полезные возможности операции **defined.** Предположим, что некоторый важный_текст должен быть передан компилятору только в том случае, если идентификатор Y определен как препроцессорный, а идентификатор N не определен. Директивы препроцессора могут быть записаны следующим образом:

\# **defined Y && !defined N**
важный_текст
#endif

Обработку препроцессор ведет следующим образом. Во-первых, определяется истинность выражений **defined** Y и **!defined** N. Получаем два значения, каждое 0L или 1L. К результатам применяется операция **&&** (конъюнкция), и при истинности ее результата важный_текст передается компилятору.

Не используя операцию **defined,** то же самое условие можно записать таким наглядным способом:

<u>Пример 10.16</u>

```
#ifdef Y
#ifndef N
важный_текст
#endif
#endif
```

Таким образом, из примера видно, что:
#if defined эквивалентно **#ifdef**
#if !defined эквивалентно **#ifndef**

Библиотека БГУИР

Стандарт языка Си не определил **defined** в качестве ключевого слова. В тексте программы его можно использовать в качестве идентификатора, свободно применяемого программистом для обозначения объектов, **defined** имеет специфическое значение только при формировании выражений-условий, проверяемых в директивах **#if** и **#elif**. В то же время идентификатор **defined** запрещено использовать в директивах **#define** и **#undef.**

8. Макроподстановки средствами препроцессора

Макрос, по определению, есть средство замены одной последовательности символов на другую. Для выполнения замен должны быть заданы соответствующие макроопределения. Простейшее макроопределение уже было введено при рассмотрении замены в тексте с помощью директивы

#define идентификатор строка_замещения

С помощью директивы **#define** можно вводить собственные обозначения базовых или производных типов. Например, директива

#define REAL long double

вводит название (имя) **REAL** для типа **long double.** Далее в тексте программы можно определять конкретные объекты, используя **REAL** в качестве обозначения их типа **(long double):**

REAL x, array[6];

Идентификатор в команде **#define** может определять, как видим, имя константы, если строка_замещения задает значение этой константы. В более общем случае идентификатор служит обозначением некоторого выражения, например:

#define RANGE ((INT_MAX) - (INT_MIN)+1)

Идентификаторы, входящие в строку замещения, в свою очередь могут быть определены как препроцессорные, и их значения будут подставлены вместо них (вместо **INT_MAX** и **INT_MIN** в нашем примере).

Допустимость выполнять с помощью **#define** «цепочки» подстановок расширяет возможности этой директивы, однако она имеет существенный недостаток – строка замещения фиксирована. Большие возможности предоставляет макроопределение с параметрами

#define имя (список параметров) строка_замещения

Здесь имя – имя макроса (идентификатор), список параметров – список разделенных запятыми идентификаторов. Между именем макроса и скобкой, открывающей список параметров, не должно быть пробелов.

Для обращения к макросу («для вызова макроса») используется конструкция («макровызов») вида

Имя_макроса (список_аргументов)

Библиотека БГУИР

В списке аргументы разделены запятыми. Каждый аргумент – препроцессорная лексема.

Классический пример макроопределения

#define max(a,b) (a < b ? b : a)

позволяет формировать в программе выражение, определяющее максимальное из двух значений аргументов. При таком определении вхождение в программу макровызова **max(X, Y)** заменяется выражением **(X < Y ? Y : X)**, а использование конструкции вида **max(Z, 4)** приведет к формированию выражения (**Z < 4 ? 4 : Z**).

В первом случае при истинном значении **X < Y** возвращается значение **Y**, иначе – значение **X**. Во втором примере значение переменной **Z** сравнивается с константой 4 и выбирается большее из значений. Не менее часто используется определение

#define ABS(X) (X<0? - (X) : X)

С его помощью в программу можно вставлять выражение для определения абсолютных значений переменных. Конструкция **ABS(E-Z)** заменяется выражением **(E-Z<0?-(E-Z):E-Z)**, в котором результат вычисления определяет абсолютное значение выражения **E-Z**. Обратите внимание на скобки. Без них могут появиться ошибки в результатах.

Следует отметить, что последовательные препроцессорные подстановки выполняются в строке замещения, но не действуют на параметры макроса.

9. Моделирование многомерных массивов

Основным понятием в языке Си является одномерный массив, а возможности формирования многомерных массивов (особенно с переменными размерами) весьма ограничены. Напомним, что нумерация элементов массивов в языке Си начинается с нуля, т.е. для обращения к начальному (первому) элементу массива требуется нулевое значение индекса.

При работе с матрицами обе указанные особенности массивов языка Си создают по крайней мере неудобства. Во-первых, при обращении к элементу матрицы нужно указывать два индекса – номер строки и номер столбца элемента матрицы. Во-вторых, нумерацию строк и столбцов матрицы принято начинать с 1.

Применение макросов для организации доступа к элементам массива позволяет программисту обойти оба указанных затруднения, правда, за счет нетрадиционных обозначений индексированных элементов (индексы в макросах, представляющих элементы массивов матриц, заключены в круглые, а не в квадратные скобки).

Рассмотрим следующую программу:

<u>Пример 10.17</u>

```
#define N    4    /*  Число строк матрицы */
```

Библиотека БГУИР

```
#define M    5     /*  Число столбцов матрицы */
#define A(i,j) x[M*(i-l) + (j-1)]
#include  <stdio.h>
void main (  )
{  /* Определение одномерного массива */
   double x[N*M];
   int i, j, k;
   for (k = 0; k < N*M; k++)
   x[k]=k;
   for (i = l; i <= N; i++) /* Перебор строк */
   {  printf (“\n Строка %d:”, i);
      /* Перебор элементов строки */
      for (j = l; j <= M; j++)
      printf<” %6.1f”, A(i, j));
}}
```

Строка 1	0.0	1.0	2.0	3.0	4.0
Строка 2	5.0	6.0	7.0	8.0	9.0
Строка 3	10.0	11.0	12.0	13.0	14.0
Строка 4	15.0	16.0	17.0	18.0	19.0

Рис. 10.2. Результат выполнения программы

В программе определен одномерный массив x[ ], количество элементов в котором зависит от значений препроцессорных идентификаторов N и M.

Значения элементам массива x[ ] присваиваются в цикле с параметром k. Далее для доступа к элементам того же массива x[ ] используются макровызовы вида A(i, j), причем i изменяется от 1 до N, а переменная j изменяется во внутреннем цикле от 1 до M. Переменная i соответствует номеру строки матрицы, а переменная j играет роль второго индекса, т.е. указывает номер столбца. При таком подходе программист оперирует с достаточно естественными обозначениями A(i, j) элементов матрицы, причем нумерация столбцов и строк начинается с 1, как и предполагается в матричном исчислении.

В тексте программы за счет макрорасширений в процессе препроцессорной обработки выполняются замены параметризованных обозначений A(i, j) на x[5*(i-l)+(j-1)], и далее действия выполняются над элементами одномерного массива x[ ]. Но этих преобразований не видно, и можно считать, что идет работа с традиционными обозначениями матричных элементов. Использованный в программе оператор (вызов функции)

printf (“% 6.lf”, A (i, j)) ;

после макроподстановок будет иметь вид

printf (“% 6.lf”, x[5*(i-l) + (j-l)]) ;

Библиотека БГУИР

На рис. 10.3 приведена иллюстративная схема одномерного массива x[ ] и виртуальной (существующей только в воображении программиста, использующего макроопределения) матрицы для рассмотренной программы.

Одномерный массив x[20]:

0	1	2	3	4	5	…	16	17	18	19
0.0	1.0	2.0	3.0	4.0	5.0	….	16.	17.	18.0	19.
M = 5							M = 5			

j-й столбец

i-я

0.0	1.0	2.0	3.0	4.0	Матрица
5.0	6.0	7.0	8.0	9.0	С
10.	11.	12.0	13.	14.	Размера
15.	16.	17.0	18.	19.	4x5

А (1, 1) соответствует x[5*(1-1)+(1-1)] = = x[0]
А (1, 2) соответствует x[5*(1-1)+(2-1)] = = x[1]
А (2, 1) соответствует x[5*(2-1)+(1-1)] = = x[5]
А (3, 4) соответствует x[5*(3-1)+(4-1)] = = x[13]

Рис. 10.3. Имитация матрицы с помощью макроопределения и одномерного массива

10. Отличия макросов от функций

Сравнивая макросы с функциями, заметим, что в отличие от функции, определение которой всегда присутствует в одном экземпляре, тексты, формируемые макросом, вставляются в программу столько раз, сколько раз используется макрос. Обратим внимание на еще одно отличие: функция определена для данных того типа, который указан в спецификации ее параметров и возвращает значение только одного конкретного типа. Макрос пригоден для обработки параметров любого типа, допустимых в выражениях, формируемых при обработке строки замещения. Тип получаемого значения зависит только от типов параметров и от самих выражений. Таким образом, макрос может заменять несколько функций. Например, приведенные выше макросы max() и ABS() верно работают для параметров любых целых и вещественных типов, а результат зависит только от типов параметров.

Для устранения неоднозначных или неверных использований макроподстановок параметры в строке замещения и ее саму полезно заключать в скобки. Еще одно отличие: фактические параметры функций – это выражения, а аргументы вызова макроса – препроцессорные лексемы, разделенные запятыми. Аргументы макрорасширениям не подвергаются.

Библиотека БГУИР

11. Препроцессорные операции в строке замещения

В последовательности лексем, образующей строку замещения, предусматривается использование двух операций – ‘#’ и ‘##’, первая из которых помещается перед параметром, а вторая – между любыми двумя лексемами. Операция ‘#’ требует, чтобы текст, замещающий данный параметр в формируемой строке, заключался в двойные кавычки.

В качестве полезного примера с операцией ‘#’ рассмотрим следующее макроопределение:

#define print (A) printf (#A”=%f”, A)

К макросу print (A) можно обращаться, подставляя вместо параметра A произвольные выражения, формирующие результаты вещественного типа.

Пример 10.18

```
print (sin (a/2)); – обращение к макросу;
printf (“sin (a/2)””=%f”, sin (a/2)); – макрорасширение.
Фрагмент программы:
double a=3.14159;
print (sin   (a/2));
Результат выполнения (на экране дисплея):
sin (a/2)=1.0
```

Операция ‘##’, допускаемая только между лексемами строки замещения, позволяет выполнять конкатенацию лексем, включаемых в строку замещения.

Чтобы пояснить роль и место операции ‘##’, рассмотрим, как будут выполняться макроподстановки в следующих трех макроопределениях с одинаковым списком параметров и одинаковыми аргументами.

Пример 10.19

```
#define zero (a, b, c, d) a (bcd)
#define one (a, b, c, d) a (b c d)
#define two (a, b, c, d) a (b##c##d)
Макровызов:              Результат макроподстановки:
zero(sin,   x,   +,   y)   sin(bcd)
one (sin,   x,   +,  y)    sin(x + y)
two(sin,   x,   +,  y)     sin(x+y)
```

В случае zero() последовательность “bcd” воспринимается как отдельный идентификатор. Замена параметров b, c, d не выполнена. В строке замещения макроса one() аргументы отделены пробелами, которые сохраняются в результате. В строке замещения для макроса two() использована операция ‘##’, что позволило выполнить конкатенацию аргументов без пробелов между ними.

Библиотека БГУИР

12. Вспомогательные директивы

В отличие от директив **#include**, **#define** и всего набора команд условной компиляции (**#if**...) рассматриваемые в данном параграфе директивы не так часто используются в практике программирования.

Препроцессорные обозначения строк

Для нумерации строк можно использовать директиву

#line константа

Она указывает компилятору, что следующая ниже строка текста имеет номер, определяемый целой десятичной константой. Директива может одновременно изменять не только номер строки, но и имя файла:

#line константа "имя файла"

Директиву #line можно «встретить» сравнительно редко, за исключением случая, когда текст программы на языке Си генерирует какой-то другой препроцессор.

Смысл директивы **#line** становится очевидным, если рассмотреть текст, который препроцессор формирует и передает на компиляцию. После препроцессорной обработки каждая строка имеет следующий вид:

имя_файла номер_строки текст_на _языке _Си

Пример 10.20

Пусть препроцессор получает для обработки файл "kkk.c" с таким текстом:

```
#define N 3 /* Определение константы */
void main() {
#line 23 "file.c"
double z[3*N]; }
```

После препроцессора в файле с именем " kkk.i" будет получен набор строк:

```
kkk.c 1:
kkk.c 2: void main()
kkk.c: 3 {
kkk.c 4:
file.c 23: double z[3*3]
file.c 24 }
```

В результирующем тексте отсутствуют препроцессорные директивы и комментарии. Соответствующие строки пусты, но включены в результирующий текст. Для них выделены порядковые номера (1 и 4). Следующая строка за директивой **#line** обозначена в соответствии со значением константы (23) и указанным именем файла "file.c".

Библиотека БГУИР

13. Реакция на ошибки

Обработка директивы

#error последовательность_лексем

приводит к выдаче диагностического сообщения в виде, определенном последовательностью лексем. Естественно применение директивы **#error** совместно с условными препроцессорными командами. Например, определив некоторую препроцессорную переменную **NAME** следующим образом:

#define NAME 5

в дальнейшем можно проверить ее значение и выдать сообщение, если у **NAME** окажется другое значение:

```
#if (NAME !=5)
#error NAME должно быть равно 5 !
```

Сообщение будет выглядеть так:

```
Error <имя_файла> <номер_строки>:
Error directive: NAME должно быть равно 5 !
```

В интегрированной среде сообщение будет выдано на этапе компиляции в следующем виде:

```
Fatal <имя_файла> <номер_строки>:
Error directive: NAME должно быть равно 5!
```

В случае выявления такой аварийной ситуации дальнейшая препроцессорная обработка исходного текста прекращается, и только та часть текста, которая предшествует условию **#if**..., попадает в выходной файл препроцессора.

14. Пустая директива

Существует директива, использование которой не вызывает никаких действий.

Она имеет следующий вид:

#

Прагмы. Директива **#pragma** – последовательность лексем, которая определяет действия, зависящие от конкретной реализации компилятора. Например, в некоторые компиляторы входит вариант этой директивы для извещения компилятора о наличии в тексте программы команд на языке ассемблера.

Возможности команды **#pragma** могут быть весьма разнообразными и важными. Стандарта для них не существует. Если конкретный препроцессор

Библиотека БГУИР

встречает прагму, которая ему неизвестна, он ее просто игнорирует, как пустую директиву. В некоторых реализациях включена прагма

#pragma pack(n),
где **n** может быть 1, 2 или 4.

Эта прагма позволяет влиять на упаковку смежных элементов в структурах и объединениях.

Соглашение может быть таким:

pack(1) – выравнивание элементов по границам байтов;
pack(2) – выравнивание элементов по границам слов;
pack(4) – выравнивание элементов по границам двойных слов.

В некоторые компиляторы включены прагмы, позволяющие изменять способ передачи параметров функциям, порядок помещения параметров в стек и т.д.

15. Встроенные (заранее определенные) макроимена

Существуют встроенные (заранее определенные) макроимена, доступные препроцессору во время обработки. Они позволяют получить следующую информацию:

LINE – десятичная константа – номер текущей обрабатываемой строки файла с программой на Си. Принято, что номер первой строки исходного файла равен 1;

FILE – строка символов – имя компилируемого файла. Имя изменяется всякий раз, когда препроцессор встречает директиву **#include** с указанием имени другого файла. Когда включения файла по команде **#include** завершаются, восстанавливается предыдущее значение макроимени **_FILE_;**

DATE – строка символов в формате: «месяц число год», определяющая дату начала обработки исходного файла;

TIME – строка символов вида «часы минуты секунды», определяющая время начала обработки препроцессором исходного файла;

STDC – константа, равная 1, если компилятор работает в соответствии с ANSI-стандартом. В противном случае значение макроимени **_STDC_** не определено. Стандарт языка Си предполагает, что наличие имени **_STDC_** определяется реализацией, так как макрос **_STDC_** относится к нововведениям стандарта.

В конкретных реализациях компиляторов набор предопределенных имен шире. Например, в препроцессор могут быть дополнительно включены:

CDECL – идентифицирует порядок передачи параметров функциям, значение 1 соответствует порядку, принятому в языке Си (в отличие от языка Паскаль);

Библиотека БГУИР

CONSOLE – определено для 32-разрядного компилятора и установлено в 1 для программ консольного приложения;

DLL – соответствует работе в режиме Windows DLL;

PASCAL – противоположен **_CDECL_** ;

WINDOWS – означает генерацию кода для Windows;

WIN32 – определен для 32-разрядного компилятора и установлен в 1 для консольных приложений.

Для получения более полных сведений о предопределенных препроцессорных именах следует обращаться к документации по конкретному компилятору.

Варианты индивидуальных заданий

Рекомендуется выполнять каждое задание, максимально используя возможности препроцессора. Ввод данных выполнять из файла. Все логически законченные части программы оформлять в виде отдельных функций. При тестировании программ продемонстрировать работу всех вариантов без удаления кода соответствующего варианта.

1. Разработать несколько вариантов программы вычисления произведения двух матриц с использованием директив препроцессора:

- для целочисленных значений;
- для вещественных значений;
- с использованием указателей.

При тестировании программы продемонстрировать работу всех вариантов без удаления кода соответствующего варианта с помощью директивы мультиветвления препроцессора.

2. Разработать программу вычисления произведения двух матриц с вещественными величинами и использованием динамического выделения памяти. Создать определения макросов, которые подсчитывают, сколько раз в программе произошло использование функций динамического выделения и освобождения памяти, а также объем выделенной и освобожденной памяти.

3. Разработать программу с использованием массива структур под названием «телефонная книжка» для хранения информации о студентах своей группы в виде: имя, фамилия, телефон. Добавление элементов в «книжку», распечатки элемента «книжки» и печати всей «книжки» целиком оформить в виде отдельных функций. Тела функций и их прототипы поместить в отдельные заголовочные файлы. Все необходимые макроопределения и именованные константы поместить в отдельный заголовочный файл. Собрать проект, максимально используя директивы препроцессора.

4. Разработать несколько вариантов программы вычисления произведения двух матриц с использованием директив препроцессора:

- для целочисленных значений;
- для вещественных значений;
- с использованием указателей.

Библиотека БГУИР

Присутствие или отсутствие вывода на экран исходной, результирующей матрицы осуществить с помощью директив условной компиляции.

5. Разработать программу вычисления произведения двух вещественных матриц переменных размеров с использованием директив препроцессора. Использовать макросы-функции для организации нетрадиционных для Си обозначений индексированных элементов, но в математическом смысле более традиционных обозначений. Доступ к элементам двумерных массивов организовать как доступ к элементам матрицы с учетом натуральной записи номера столбца (от 1 до N) и номера строки матрицы (от 1 до M) в виде A(i, j).

6. Разработать программу с использованием препроцессорных средств:

- имеется одномерный массив, в котором расположены по порядку натуральные числа от 1 до 100;
- из элементов одномерного массива построить все возможные двумерные массивы по возрастанию числа строк, в которых надо просуммировать все элементы, стоящие на главной диагонали соответствующей матрицы;
- доступ к элементам двумерных массивов организовать как доступ к элементам матрицы с учетом натуральной записи номера столбца (от 1 до N) и номера строки матрицы (от 1 до M) в виде A(i, j).

Максимально использовать макроопределения-функции как для организации доступа к элементам, так и для организации печати результата. Использовать прием создания строк из аргументов макроса с помощь операции препроцессора #.

7. Создать одномерный массив, состоящий из вещественных чисел. Разработайте программу, которая с помощью макроса-функции разделяет массив на два массива с четными и нечетными числами. Создать определение макроса-функции, которая возвращает значение 1, если переменная X представляет число (не обязательно целочисленное) четное и больше числа Y.

8. Создать определение макроса-функции, которая выводит на печать представления и значения двух целочисленных значений. Например, он может выводить данные в виде:

3 + 4 = 7, 4*12 = 48,

если аргументами служат выражения 3 + 4 и 4 * 12.

9. Создать определение макроса, который печатает имя, значение и адрес переменной типа int в следующем формате:

Имя: fop;   значение: 23;   адрес: ff46016.

10. Создать определение макроса-функции, которая в массиве (необязательно целочисленном) находит заданное значение и возвращает его. С помощью этого макроса-функции разработайте программу упорядочения массива по возрастанию.

11. Гармоническое среднее двух чисел получают вычислением среднего от инверсий этих чисел с последующим инвертированием результата. Воспользоваться директивой **#define** для определения макроса-функции,

Библиотека БГУИР

который выполняет эту операцию, и написать простую программу для тестирования этого макроса.

12. Написать программу форматирования текста, читаемого из файла и состоящего из строк ограниченной длины. Слова разделены произвольным количеством пробелов. Программа должна читать файл по строкам, форматировать каждую строку и выводить строку в выходной файл. Форматирование заключается в удалении лишних пробелов и выравнивании границы текста слева. Кроме того, программа должна подсчитать число строк в тексте. Программу разместить в двух исходных файлах и одном заголовочном файле, предусмотрев при этом защиту от повторного включения заголовочных файлов.

13. В полярной системе координат вектор описывается модулем и углом с осью x в направлении против часовой стрелки. В прямоугольной системе координат тот же вектор описывается составляющими x и y, как показано на рис. 10.4. Написать программу, которая считывает значения модуля и угла (в градусах) вектора, а затем отображает составляющие x и y. При этом воспользоваться следующими уравнениями:

$x = r \cos A, \; y = r \sin A.$

Для выполнения преобразования применить макрос-функцию, которая принимает структуру, содержащую полярные координаты, и возвращает структуру, содержащую прямоугольные координаты (можно воспользоваться указателями на эти структуры).

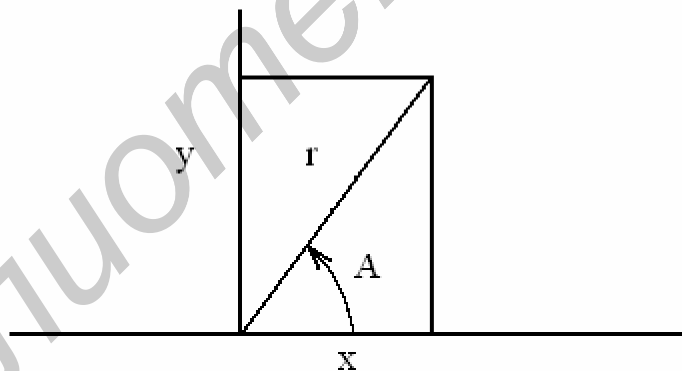


Рис. 10.4. Прямоугольные и полярные координаты

14. Библиотека ANSI содержит функцию clock () со следующим описанием:

```
#include <time.h>
clock_t clock   (void);
```

Здесь clock_t – тип данных, определенный в файле time.h. Функция возвращает процессорное время в единицах, которые зависят от реализации языка. (Если процессорное время недоступно или не может быть представлено, функция возвращает значение –1.) Однако в файле time.h также определена константа CLOCKS_PER_SEC, которая представляет количество единиц процессорного времени в секунду. Следовательно, в результате деления

Библиотека БГУИР

разницы между двумя возвращаемыми значениями функции clock () на константу CLOCKS_PER_SEC получается количество секунд, прошедшее между двумя вызовами функции. Приведение значений к типу double до операции деления позволит получать результат в долях секунды. Написать функцию, которая принимает аргумент типа double, представляющий промежуток времени, а затем выполняет цикл до истечения указанного периода времени. Написать простую программу для тестирования этой функции.

15. Написать макрос-функцию, которая в качестве аргумента принимает имя массива элементов типа int, размер массива и значение, представляющее количество выборок. Функция должна случайным образом выбирать из массива указанное количество элементов и печатать их значения. Ни один элемент массива не должен выбираться более одного раза. (Это модель выбора чисел в лотерее или членов жюри.) Воспользоваться функцией srand(), чтобы инициализировать rand() – генератор случайных чисел.

16. Разработать программу, которая определяет, в каком из двух целочисленных произвольных массивов больше положительных элементов. Создать определение макроса-функции, которая в массиве (необязательно целочисленном) находит заданное значение и возвращает его. Все константы определить через макроопределения.

17. Разработать программу с двумя типами данных «точка» и «прямоугольник», причем второй тип определяется через первый тип. Каждый тип данных и функции работы с ним описываются в отдельном заголовочном файле. В основном модуле просто создается объект типа «прямоугольник» и выводятся координаты его левого верхнего и правого нижнего углов. В основном модуле используются как функции из одного заголовочного файла, так и из другого. С использованием директив условной компиляции предотвратить повторное включение заголовочных файлов.

18. Написать программу, которая для целочисленного массива из 10 элементов определяет, сколько положительных элементов располагается между его максимальным и минимальным элементами. Создать определение макроса-функции, которая в массиве (необязательно целочисленном) находит заданное значение и возвращает его. Все константы определить через макроопределения.

19. Разработать программу вычисления и вывода таблицы умножения для чисел от 1 до 10 так, как это предлагается в школьных тетрадях по арифметике с использованием макросов-функций и препроцессорных операций в строке замещения.

20. Использовать директивы условной компиляции для того, чтобы выводить только один столбец таблицы умножения, без удаления кодов других частей программы из задачи 19.

21. Разработать программу вывода одного столбца календаря для текущего месяца с указанием дней недели с использованием макросов-функций и препроцессорных операций в строке замещения.

Библиотека БГУИР

22. Разработать как многофайловый проект программу вывода календаря на 2007 год с указанием дней недели с использованием макросов-функций и препроцессорных операций в строке замещения.

23. В одномерном массиве, состоящем из n вещественных элементов, вычислить:

- сумму положительных элементов массива;
- произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.

Упорядочить элементы массива по убыванию. Использовать макроопределения и макросы-функции.

24. Имеется три вложенные друг в друга фигуры: в квадрат встроен круг, в круг встроен равносторонний треугольник. Расположить по убыванию размеры площадей этих фигур. Все оформить в виде макросов-функций. Для вывода результата также использовать макроопределения и препроцессорные операции # и # #.

ЛАБОРАТОРНАЯ РАБОТА №11

ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ПАМЯТИ

Цель: Изучить способы распределения памяти в программах на языке Си, функции библиотеки Си для динамического распределения памяти, операторы new и delete языка С++. Разобрать приведенные примеры и выполнить один или более вариантов заданий.

Теоретические сведения

Память для хранения данных может выделяться статически и динамически. Статическое распределение памяти выполняется на этапе компиляции. Компилятор по имени и типу объектов отводит для них место в результирующем exe-файле в разделе сегмента данных (или стеке).

Чаще при программировании неизвестно, какого размера массивы, структуры или другие данные будут необходимы для работы программы. В этом случае необходимо для хранения данных использовать специальный раздел памяти, называемый «кучей» (heap). Куча располагается вне любой программы, объём кучи и её месторасположение зависят от модели памяти.

В библиотеке функций языка Си существует ряд функций, выполняющих динамическое распределение памяти. Основные функции представлены в табл. 11.1.

Библиотека БГУИР

Таблица 11.1

Функции библиотеки Си для динамического распределения памяти

Функция распределения	Функция освобождения	Функция перераспределения
Malloc	Free	Realloc
Calloc	Free	Realloc

Функция malloc запрашивает у операционной системы определенное число байт памяти, функция calloc отличается от malloc параметрами, она запрашивает у операционной системы блок из некоторого числа элементов определенного размера. Функция realloc меняет разделы выделенного блока.

Адрес начала выделенной памяти возвращается в точку вызова функции. Если выделенный участок памяти больше не требуется, он должен быть освобождён с помощью функции free в том же блоке, где и запрошен, иначе эта память оказывается недоступной для дальнейшего распределения.

1. Функции malloc и free

Формат malloc:

указатель = (тип) malloc (количество байт);

Формат free:

free(указатель);

Пример 11.1

Выполнить распределение и освобождение памяти для двух чисел типа int, и адрес начала поместить в указатель num.

```
int * num;            //объявление указателя
num=(int*) malloc(2);    //выделение памяти
free(num);              //освобождение памяти
```

Пример 11.2

Выделить память под строку из 20 символов, инициализировать ее и вывести на экран. Освободить память.

```
#include<stdio.h>
#include<string.h>
# include <stdlib.h>
void  main(void)
{    char *stroka;
```

Библиотека БГУИР

```
    stroka=(char *) malloc(20);    //выделение памяти
    strcpy(stroka,"Good luck!");   //инициализация строки
    printf("%s\n", stroka);        //вывод на экран
    getchar();
    free(stroka);                  //освобождение памяти
}
```

Пример 11.3

Выделить память под структуру student, состоящую из 3-х полей: массива фамилий, массива адресов e-mail, массива номеров телефонов.

```
#include<stdlib.h>
struct student
{     char name[40];
      char mail[40];
      char phone[40];
};
struct student *get_mem(void)
{       struct student * sp;
        sp=(struct student *)malloc(sizeof(student));
        if(!sp)
        { printf(“Allocation error.”);   exit(1);  }
         return sp;
}
void main(void)
{     struct student * Alisa;
      Alisa = get_mem();
      strcpy(Alisa ->name,"Alisa");
      strcpy(Alisa -> mail,"Alisa@tut.by ");
      strcpy(Alisa -> phone,"666 66 66");
      printf(“%s\n”, Alisa ->mail);
      getchar();    }
```

В настоящем примере проводится проверка корректности выделения памяти для указателя sp. Если память не выделена, то значение указателя равно нулю и следует проанализировать эту ситуацию и предусмотреть аварийную обработку.

Пример 11.4

Задать массив из 100 строк через указатель. Зарезервировать память 128 символов под каждую строку массива. Освободить зарезервированную память.

Библиотека БГУИР

```
int main(void)
{
      char *str[100];
      int i;
      for(i=0; i<100; i++)
      {
            str[i]=(char*)malloc(128);
            if(str[i]==NULL)
           {
            printf("Alocation error."); exit(1);
            }
       }
  for(i=0; i<100; i++) free(str[i]);          //освобождение памяти
  return 0;
}
```

2. Операторы new и delete

Формат:

Переменная-указатель = new тип_переменной [размер];

размер определяет число элементов массива.

Ограничения при размещении массива:

- его нельзя инициализировать;
- для освобождения динамически размещённого массива необходимо использовать следующую форму оператора: delete[] переменная-указатель; квадратные скобки информируют оператор delete, что необходимо освободить память, выделенную для массива.

<u>Пример 11.5</u>

Выделяется память для массива из 10 элементов типа float со значениями элементов от 100 до 109, а затем содержимое массива выводится на экран.

```
#include<iostream.h>
#include<escept.h>
int main()   {
      float *p;
      int i;
      try
      {   p=new float[10];              // получение 10 элементов    }
      catch (xalloc xa)
      {      cout <<"размещение невозможно\n";
```

Библиотека БГУИР

```
            return 1;    }
      for(i=0;i<10;i++)               //присваивание значений от 100 до 109
      p[i]=100.00+i;
      delete []p;                       //удаление всего массива
      return 0;
}
```

Пример 11.6

```
# include <stdio.h>
int main(void)
{
      char* s;
      int t;
      s=new char[80];
      printf ("Vvedite stroku\n");
      gets(s);
      for(t=80;t>=0;t--)
      printf("%c",s[t]);
      delete s;
      return 0;
}
```

Данная программа показывает способ использования динамически выделенного массива для чтения символов с клавиатуры с помощью функции gets().

Для языка С++ реализованы операторы new – захвата памяти и delete – освобождения памяти, доступные пользователю в консольном режиме среды Visual C++.

Формат операторов:

Указатель = new тип (значение);

delete указатель;

Пример 11.7

Распределить память для целочисленной переменной data и инициализировать значением 10.

```
int *data;
data = new int(10);
delete data;
```


Библиотека БГУИР

Пример 11.8

Распределить память для массива целочисленных переменных X1 размерностью 80.

```
int *X1;
X1 = new int [80];
delete []X1;
```

Память для массива освобождается с помощью оператора delete[]. С помощью new могут быть размещены любые типы данных. Массив нельзя инициализировать с помощью оператора new.

Пример 11.9

Распределить память для массива из 10 элементов типа float со значениями элементов от 100 до 109, а затем содержимое массива вывести на экран.

```
#include<iostream.h>
int main()
{      float *X1;    int i;
       try
       {      X1=new float[10];        // распределить 10 элементов    }
       catch (char * code)
       {      cout <<"Alocation error\n";
              return 1;
       }
       for(i=0; i<10; i++)  //присваивание значений от 100 до 109
       X1[i]=100.00+i;
       printf("Array X1:\n");
       for(i=0; i<10; i++)
       printf("%f\n", X1[i])  // печать очередного элемента
       getchar();
       delete []X1;             // удаление всего массива
       return 0;
}
```

В данной программе блок try/catch отслеживает ошибки выделения памяти. Если внутри блока try возникает прерывание, управление передается в блок catch. Если прерывание не возникло, блок catch не выполняется.

В этом примере используются операторы ввода-вывода стандартных типов данных и строк:

операция cout << (операция передачи в стандартный выходной поток cout);

Библиотека БГУИР

операция cin >> (операция извлечения из стандартного входного потока cin).

Пример 11.10

```
cout<<"Vvedite chislo";
cin>>n;
```

Это операции языка C++. Они не требуют форматирующих строк и спецификаторов преобразования для указания на тип входных и выходных данных, а также при использовании с операцией извлечения из потока переменной n не предшествует операция взятия адреса &.

Пример 11.11

Распределить память для массива из 80 символов. Прочитать в эту память строку с клавиатуры. Вывести полученный массив символов на экран в обратном порядке. Освободить память.

```
# include <stdio.h>
int main(void)
{      char* string;  int t;
       string =new char[80];
       if (string ==0)
       {
              printf ("Alocation error\n");
              exit(1);
       }
       printf ("Vvedite stroku\n");
       gets(string);
       for(t=80; t>=0; t--)
       printf("%c", string [t]);
       getchar();
       delete []string;
       return 0;
}
```

В данном примере для контроля корректности распределения памяти используется проверка указателя string на нуль. Если указатель равен нулю, то выделение памяти выполнено некорректно и следует прекратить выполнение программы или выполнить какие-либо другие аварийные действия.

Библиотека БГУИР

Данная программа показывает способ использования динамически выделенного массива для чтения символов с клавиатуры с помощью функции gets().

Часто операторы new и delete используются для работы с массивами, длина которых заранее неизвестна. Например, если количество элементов массива определяется в программе непосредственно перед использованием массива.

<u>Пример 11.12</u>

Подсчитать сумму элементов массива переменной длины. Как правило, перед вводом значений элементов массива пользователю предлагается ввести количество элементов массива, а затем сами элементы массива:

```
#include <stdio.h>
void main(void)
{
     int *a;
     int k;
     int i;
     printf("Vvedite kolichestvo elementov massiva:\n");
     scanf("%d",&k);
     a = new int[k];         // выделение памяти под массив
     for(i=0;i<k;i++)        // ввод элементов массива
     scanf("%d",&a[i]);
     int sum=0;
     for(i=0;i<k;i++)  // расчет суммы элементов массива
     sum=sum+a[i]);
     printf("%d",sum);     // вывод суммы элементов массива
     getchar();
     delete []a;          // освобождение памяти
}
```

Варианты индивидуальных заданий

1. Задать массив структур student. Распределить память для элементов массива с помощью функции get_mem. Инициализировать 5 элементов массива константами. Вывести массив структур на экран. Освободить память.

2. Задать массив структур student. Распределить память для элементов массива с помощью функции get_mem. Инициализировать 5 элементов массива путем ввода информации с клавиатуры (scanf). Вывести массив структур на экран. Освободить память.

Библиотека БГУИР

3. Задать массив структур student. Распределить память для элементов массива с помощью функции get_mem. Инициализировать 10 элементов массива константной информацией (поле ФИО). Остальную информацию (номер телефона, e-mail) вводить с клавиатуры (scanf). Вывести массив структур на экран. Освободить память.

4. Задать 2 массива структур student1 и student2. Распределить память для элементов массивов с помощью функции get_mem. Инициализировать элементы массивов путем ввода информации с клавиатуры (scanf). Первым вводится с клавиатуры номер группы. Далее для группы 1 информация попадает в массив student1, для группы 2 – в массив student2. Вывести оба массива структур на экран. Освободить память.

5. Задать 2 массива структур student1 и student2. Распределить память для элементов массивов с помощью функции get_mem. Инициализировать элементы массивов путем ввода информации с клавиатуры (scanf). Первым вводится с клавиатуры номер группы. Далее для группы 1 информация попадает в массив student1, для группы 2 – в массив student2. Вывести из обоих массивов на экран информацию обо всех студентах и их телефонах. Освободить память.

6. Задать 2 массива структур student1 и student2. Распределить память для элементов массивов с помощью функции get_mem. Инициализировать элементы массивов путем ввода информации с клавиатуры (scanf). Первым вводится с клавиатуры номер группы. Далее для группы 1 информация попадает в массив student1, для группы 2 – в массив student2. Вывести из обоих массивов на экран информацию обо всех студентах и их e-mail. Освободить память.

7. Задать 2 массива структур student1 и student2. Распределить память для элементов массивов с помощью функции get_mem. Инициализировать элементы массивов путем ввода информации с клавиатуры (scanf). Первым вводится с клавиатуры номер группы. Далее для группы 1 информация попадает в массив student1, для группы 2 – в массив student2. Вывести из обоих массивов на экран информацию обо всех студентах со средним баллом выше 8. Освободить память.

8. Задать 2 массива структур student1 и student2. Распределить память для элементов массивов с помощью функции get_mem. Инициализировать элементы массивов путем ввода информации с клавиатуры (scanf). Первым вводится с клавиатуры номер группы. Далее для группы 1 информация попадает в массив student1, для группы 2 – в массив student2. Вывести из обоих массивов на экран информацию обо всех студентах со средним баллом ниже 5. Освободить память.

Библиотека БГУИР

9. Задать массив структур «ведомость» (сдачи экзамена). Распределить память для элементов массива. Инициализировать элементы массива путем ввода информации с клавиатуры (scanf). Вывести массив структур на экран. Освободить память.

10. Задать массив структур «ведомость» (сдачи экзамена). Распределить память для элементов массива. Инициализировать 10 элементов массива константной информацией (поля ФИО, номер зачетки). Остальную информацию (дата сдачи, оценка) вводить с клавиатуры (scanf). Вывести массив структур на экран. Освободить память.

11. Задать массив структур «ведомость» (сдачи экзамена). Распределить память для элементов массива. Инициализировать 10 элементов массива константной информацией (поля ФИО, номер зачетки). Остальную информацию (дата сдачи, оценка) вводить с клавиатуры (scanf). Вывести на экран список студентов, получивших неудовлетворительные оценки. Освободить память.

12. Задать массив структур «ведомость» (сдачи экзамена). Распределить память для элементов массива. Инициализировать 10 элементов массива константной информацией (поля ФИО, номер зачетки). Остальную информацию (дата сдачи, оценка) вводить с клавиатуры (scanf). Вывести на экран отчет о количестве каждой оценки в ведомости. Освободить память.

13. Задать 2 массива структур «ведомость» (сдачи экзамена) для группы 1 и группы 2. Распределить память для элементов массивов. Инициализировать по 10 элементов массивов константной информацией (поля ФИО, номер зачетки). Остальную информацию (дата сдачи, оценка) вводить с клавиатуры (scanf). Вывести на экран отчет о количестве каждой оценки по двум группам. Освободить память.

14. Задать 2 массива структур «ведомость» (сдачи экзамена) для группы 1 и группы 2. Распределить память для элементов массивов. Инициализировать по 10 элементов массивов константной информацией (поля ФИО, номер зачетки). Остальную информацию (дата сдачи, оценка) вводить с клавиатуры (scanf). Вывести на экран список студентов из обоих групп, получивших неудовлетворительные оценки. Освободить память.

15. Задать 2 массива структур «ведомость» (сдачи экзамена) для разных предметов, но для одной группы. Распределить память для элементов массивов. Инициализировать по 10 элементов массивов константной информацией (поля ФИО, номер зачетки). Остальную информацию (дата сдачи, оценка) вводить с клавиатуры (scanf). Вывести на экран список студентов, получивших неудовлетворительные оценки по двум предметам. Освободить память.

Библиотека БГУИР

16. Задать массив структур «изучаемые дисциплины». Распределить память для элементов массива. Инициализировать элементы массива путем ввода информации с клавиатуры (scanf). Вывести массив структур на экран. Освободить память.

17. Задать массив структур «изучаемые дисциплины». Распределить память для элементов массива. Инициализировать элементы массива путем ввода информации с клавиатуры (scanf). Вывести на экран перечни дисциплин, читаемых каждым преподавателем. Освободить память.

18. Задать массив структур «изучаемые дисциплины». Распределить память для элементов массива. Инициализировать элементы массива путем ввода информации с клавиатуры (scanf). Вывести на экран перечни дисциплин, читаемых в каждом семестре. Освободить память.

19. Задать массив структур «изучаемые дисциплины». Распределить память для элементов массива. Инициализировать 10 элементов массива константной информацией (поле «наименование дисциплины»). Остальную информацию (поля «номер семестра», «фамилия преподавателя») вводить с клавиатуры (scanf). Вывести на экран полученный массив структур. Освободить память.

20. Задать массив структур «изучаемые дисциплины». Распределить память для элементов массива. Инициализировать элементы массива путем ввода информации с клавиатуры (scanf). Вывести на экран перечень дисциплин, читаемых в первом семестре. Освободить память.

21. Задать массив структур «изучаемые дисциплины». Распределить память для элементов массива. Инициализировать 10 элементов массива константной информацией (поле «наименование дисциплины»). Остальную информацию (поля «номер семестра», «фамилия преподавателя») вводить с клавиатуры (scanf). Вывести на экран перечень дисциплин, читаемых в первом семестре. Освободить память.

22. Создать массив из 20 строк с динамическим распределением памяти. Инициализировать массив последовательностью натуральных чисел. Вывести массив на экран построчно в обратном порядке. Освободить память.

23. Задать массив из 10 строк. Распределить динамическую память для массива строк. Инициализировать массив информацией, введенной с клавиатуры. Вывести массив на экран построчно. Освободить память.

24. Определить динамический массив целых положительных чисел. Ввести числа в массив с клавиатуры. Вывести все четные числа на экран. Освободить память.

25. Определить динамический массив целых чисел. Ввести числа в массив с клавиатуры. Вывести все отрицательные числа на экран. Освободить память.

Библиотека БГУИР

26. Определить динамический массив целых чисел. Ввести числа в массив с клавиатуры. Определить минимальное число и вывести его на экран. Освободить память.

27. Определить динамический массив целых чисел. Ввести числа в массив с клавиатуры. Определить максимальное число и вывести его на экран. Освободить память.

28. Определить динамический массив целых чисел. Ввести числа в массив с клавиатуры. Вычислить среднее арифметическое массива без учета максимального и минимального элементов. Освободить память.

29. Определить динамический массив целых чисел. Ввести числа в массив с клавиатуры. Вычислить, сколько раз в массиве встречается число, введенное с клавиатуры. Освободить память.

30. Определить динамический массив целых чисел. Ввести числа в массив с клавиатуры. Вычислить индекс в массиве числа, введенного с клавиатуры. Освободить память.

31. Написать программу ввода динамического массива (с помощью new) целых положительных чисел и распечатки на экране тех чисел, которые делятся без остатка на 3.

32. Вероятность того, что в указанный морской порт в день прибывает k кораблей, составляет P()=(3^k)*(e^3)/(k!). Используя оператор new(), написать программу для вычисления данной вероятности.

ЛАБОРАТОРНАЯ РАБОТА №12

СТРУКТУРА ДАННЫХ «СПИСОК»

Цель: Изучить структуру данных «список». Рассмотреть внутреннюю структуру, виды списков, основные операции над списками. Разобрать приведенные примеры и выполнить самостоятельно задания.

Теоретические сведения

1. Основные определения

Список – это совокупность объектов или элементов, в которой каждый объект содержит информацию о местоположении связанного с ним другого объекта.

Если список располагается в оперативной памяти, то, как правило, информация для поиска следующего объекта – это указатель, адрес памяти. Если связный список хранится на диске в файле, то информация о следующем

Библиотека БГУИР

элементе может включать смещение элемента от начала файла к положению указателя записи или считывания файла, ключ записи и любую другую информацию, позволяющую однозначно отыскать следующий элемент списка.

В списке элементы связаны друг с другом логически. Логический порядок следования элементов списка определяется с помощью указателей. Подчеркнем, что логический порядок следования элементов списка может не совпадать с физическим порядком их расположения в памяти ПЭВМ.

Списки бывают линейными и кольцевыми, односвязными и двусвязными.

Как правило, элемент списка представляет собой структурную переменную, содержащую указатель или указатели на следующий элемент и любое число других полей, называемых информационными.

Если движение от элемента к элементу списка возможно только в одном направлении и список имеет начальную точку такого движения, говорят об односвязном списке. Элемент односвязного списка включает только указатель на следующий элемент. Сам список характеризуется указателем на начало списка (рис. 12.1).

Двусвязный список позволяет выполнять «движение» от элемента к элементу в обоих направлениях. В этом случае элемент включает два указателя: на предыдущий и последующий элементы списка. А так как список имеет и начало, и конец, описываются еще два указателя – начала и конца списка (рис. 12.2).

Рис. 12.1. Модель односвязного линейного списка

Список, в котором последний элемент не связан с первым, называется линейным. Соответственно кольцевым называется список, у которого последний элемент указывает на первый. В последнем элементе односвязного и двусвязного линейного списка указатель на следующий элемент и в первом элементе двусвязного списка указатель на предыдущий элемент равны нулю.

Библиотека БГУИР

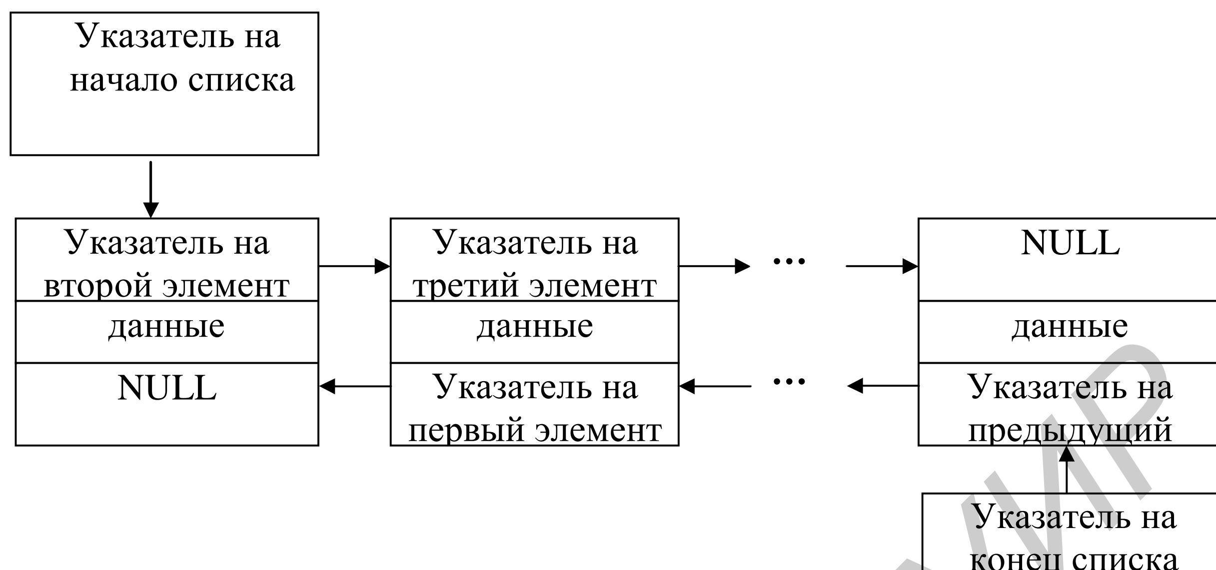


Рис. 12.2. Модель двусвязного линейного списка

2. Операции над списками

Основными операциями, которые выполняются над списками, являются перебор элементов списка, поиск заданного элемента, вставка в список нового элемента, удаление элемента из списка. Выполнение этих операций основано на изменении указателей.

2.1. Перебор элементов списка

Эта операция выполняется для линейных списков очень часто и состоит в последовательном доступе к элементам списка – ко всем до конца списка либо до нахождения искомого элемента. Для каждого из перебираемых элементов осуществляется некоторая обработка его информационной части: сравнение с образцом, печать, модификация и прочее.

2.2. Вставка элемента в список

Схематически выполнение этой операции представлено на рис. 12.3.

Как следует из рис. 12.3, указатель элемента Эл2 теперь указывает не на элемент Эл3, а на элемент Эл5. Указатель элемента Эл5 указывает на элемент Эл3. Логическая последовательность элементов будет Эл1, Эл2, Эл5, Эл3, Эл4. Подчеркнем еще раз, что в списке новый элемент Эл5 физически не обязательно помещается за элементом Эл2. Достаточно, чтобы он следовал за ним логически.

Вставка элемента в начало односвязного списка показана на рис. 12.4. В данном случае указатель переустанавливается на начало списка на вставленный элемент Эл3, а элемент Эл3 указывает на бывший первый элемент.

Библиотека БГУИР

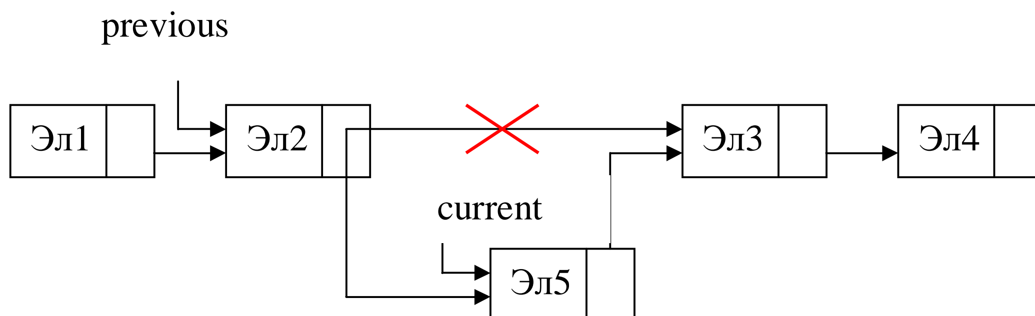


Рис. 12.3. Вставка элемента в середину односвязного списка

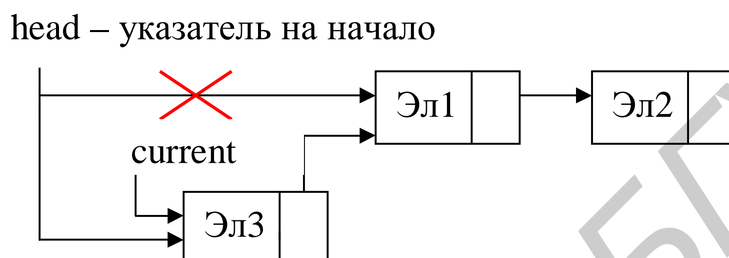


Рис. 12.4. Вставка элемента в начало односвязного списка

2.3. Удаление элемента из списка

При удалении элемента Эл3 из списка прежде всего выполняется поиск этого элемента в списке, а затем его удаление. Результат удаления показан на рис. 12.5.

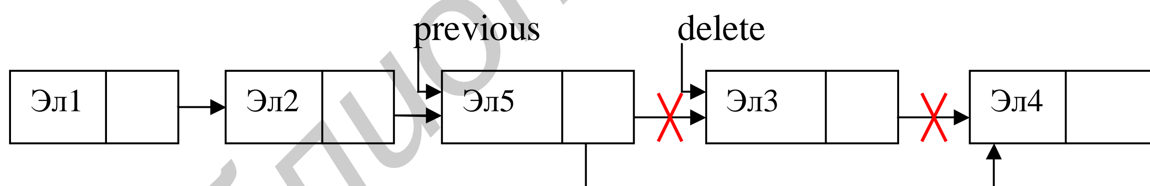


Рис. 12.5. Удаление элемента из списка

Теперь за элементом Эл5 следует элемент Эл4. Соответствующим образом изменился указатель элемента Эл5. Элемент Эл3 в списке теперь отсутствует, так как при просмотре списка, переходя от элемента к элементу в соответствии с указателями, на элемент Эл3 не попадаем. Следует отметить, что удаление элемента из списка и удаление из памяти – это не одно и то же. Элемент Эл3 будет находиться в памяти до тех пор, пока не удалится явно с помощью оператора delete. Удаление элемента из начала списка показано на рис. 12.6.

Библиотека БГУИР

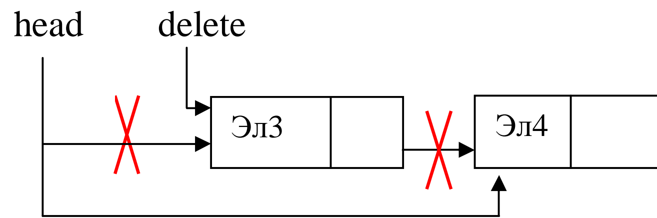


Рис. 12.6. Удаление элемента из начала списка

3. Пример реализации односвязного списка с помощью массива структур

В рассматриваемом примере используется следующая структура:

```
struct list
{       char book;
        list *next;    };
```

Предполагается, что память, отводимая под элемент списка, выделяется динамически, поэтому при реализации списков его длина ограничивается только доступным объёмом памяти. Информационное поле представляет собой символьную переменную по имени book. Признаком последующего элемента списка является наличие поля next, а признаком последнего элемента списка – равенство на NULL этого поля.

Следующая программа реализует операции:

- создает пустой список;
- добавляет элементы в список в алфавитном порядке;
- удаляет элементы из списка;
- проверяет, является ли список пустым;
- выводит список на экран.

Функция main организует запуск программы instructions для ввода кода необходимой операции над списком и вызывает функцию, которая выполняет эту операцию.

Операцию добавления элемента в список в алфавитном порядке выполняет функция insert, операцию удаления – функция del, операцию проверки, является ли список пустым, – IsEmpty, вывод списка на экран – printList.

Пример 12.1

```
#include "stdafx.h"
# include <stdio.h>
# include <stdlib.h>
```

Библиотека БГУИР

```
struct list
{
      char book;
      struct list *next;
};
typedef struct list ListNode;
typedef ListNode *ListNodePtr;
void insert (ListNodePtr*, char);
char del (ListNodePtr*, char);
int IsEmpty (ListNodePtr);
void printList (ListNodePtr);
void instructions (void);
int main ()
{     ListNodePtr start = NULL;
      int choice;
      char elem;
      instructions();                 // вывести меню
      printf("?");
      scanf("%d", &choice);
      while (choice !=3)
      {
      switch (choice)
      {
            case 1:                   /*Добавить значение в список*/
            printf ("Enter an character: ");
            scanf("\n%c", &elem);
            insert(&start, elem);
            printList (start);
            break;
            case 2:                   /*Удалить значение из списка*/
            if (!IsEmpty(start))
            {     printf("Enter character to be deleted:");
                  scanf("\n%c", &elem);
                  if (del(&start,elem))
                  {     printf("%c deleted.\n", elem);
                        printList(start);  }
                  else
                  printf("%c not found.\n", elem);
            }
```


Библиотека БГУИР

```
            else
            printf("List is empty.\n");
            break;
            default:
            printf ("Invalid choice.\n");
            instructions();
            break;
        }
        printf("?");
        scanf("%d", &choice);
        }
        printf("End of run.\n");
        return 0;
}
/*Вывести инструкции*/
void instructions(void)
{       printf("Enter choice:\n"
        "1  insert an element into the list.\n"
        "2  delete an element from the list.\n"
        "3  end program.\n");
}
/*Вставить в список новое значение в алфавитном порядке */
void insert (ListNodePtr *s, char value)
{      ListNodePtr newP, previous, current;
        NewP = (ListNodePtr) malloc (sizeof(ListNode));
        if (newP!=NULL)
        {     /*есть ли место ?*/
            newP->book =value;
            newP->next=NULL;
            previous=NULL;
            current=*s;
            while (current!=NULL && value >current->book)
            {     previous=current;
                  current=current->next;    }
            if(previous==NULL)
            {      newP->next=*s;
                   *s=newP;            }
            else
            {      previous->next = newP;
```

Библиотека БГУИР

```
                    newP->next = current;      }
          }
          else
          printf("%c not inserted. No memory available.\n", value);
}
/*Удалить элемент списка*/
char del (ListNodePtr *s, char value)
{     ListNodePtr previous, current, temp;
          if (value == (*s)->book)
          {      temp=*s;
                 *s=(*s)->next;  // отсоединить узел
                 free(temp);      // освободить отсоединенный узел
                 return value;
          }
          else
          {     previous=*s;
                 current=(*s)->next;
                 while (current!=NULL && current->book!= value)
                 {   previous=current;           /*перейти.......*/
                       current=current->next;    /*...к следующему*/
                 }
                 if (current!=NULL)
                 {      temp=current;
                        previous->next=current->next;
                        free(temp);
                        return value;    }
          }
          return '\0';
}
/*Возвратить 1, если список пуст, 0 в противном случае*/
int IsEmpty(ListNodePtr s)
{
return s==NULL;
}
/*Распечатать список*/
void printList (ListNodePtr current)
{
          if (current==NULL)
          printf("The list is empty.\n");
```


Библиотека БГУИР

```
    else
    {   printf("The list is :\n");
        while (current!=NULL)
        {   printf("%c-->",current->book);
            current=current->next;  }
    printf("NULL\n");
    }
}
```

4. Пример реализации двусвязного списка с помощью массива данных

Список можно представить в виде массива. Для представления однонаправленного списка, элементами которого являются целые числа, можно использовать двумерный массив Sp[2][MaxEl], где MaxEl равно количеству элементов списка. Причем в Sp[1, i] располагается элемент списка, а Sp[2, i] является указателем и определяет позицию следующего элемента.

Двунаправленный список можно представить двумерным массивом Sp[3][N]. В этом случае, например, можно считать, что элемент списка расположен в Sp[1][i], а Sp[2][i] и Sp[3][i] соответственно указывают предыдущий и следующий элементы списка.

Организацию списков с помощью массива данных можно выполнить различными путями. Можно зарезервировать несколько массивов одинаковой размерности. В основном массиве хранится значащая информация, во вспомогательных – значения индексов следующих элементов списка и значения индексов предыдущих элементов списка, если список двусвязный. В нижеследующем примере основная информация представлена в массиве list, индексы следующих элементов – в массиве next, индексы предыдущих элементов – в массиве prev.

Пример 12.2

```
/* Создать список с помощью массива целых чисел
Элементы списка в обратном порядке вывести на экран */
#include <stdio.h>
#include <stdlib.h>
#define MAX 50
int main (void)
{
    int list[MAX];
    int next[MAX];
    int prev[MAX];
```


Библиотека БГУИР

```
int end=0;
int begin=0;
for (int i=0; i<MAX; i++)
{ list[i]=next[i]=prev[i]=0;   }
printf("Input the number of elements: ");
int n=0;
scanf("%d",&n);
prev[0]=-1;
printf("Input elements of list: ");
int count=0;
for (i=0; i<n; i++)
{  scanf("%d",&list[i]);
       next[i]=i+1;
       prev[i]=i-1;
       count++;
}
prev[0]=-1;
next[count]=-1;
begin=0;
end=count-1;
printf("Elements of list from the end to begin: ");
int temp=end;
do {     printf("%d ",list[temp]);
             temp=prev[temp];
     } while (prev[temp]!=-1);
printf("%d \n",list[temp]);
system("PAUSE");
return 0;   }
```

Варианты индивидуальных заданий

1. Создать список с помощью массива целых чисел. Элементы списка в обратном порядке вывести на экран.

2. Создать список с помощью массива целых чисел. Все четные элементы списка вывести на экран.

3. Создать список с помощью массива целых чисел. Все нечетные элементы списка вывести на экран.

4. Создать односвязный список с помощью массива целых чисел. Отсортировать элементы списка по возрастанию, задавая порядок чисел

массивом индексов следующих элементов (next). В результате массив чисел остается без изменений, массив индексов переупорядочивается. Результирующий список вывести на экран.

5. Создать односвязный список с помощью массива целых чисел. Расположить в начале списка все четные элементы списка, задавая порядок чисел массивом индексов следующих элементов (next). В результате массив чисел остается без изменений, массив индексов переупорядочивается. Результирующий список вывести на экран.

6. Создать односвязный список с помощью массива целых чисел. Исключить из списка все нулевые элементы, задавая порядок чисел массивом индексов следующих элементов (next). В результате массив чисел остается без изменений, массив индексов переупорядочивается. Результирующий список вывести на экран.

7. Создать односвязный список с помощью массива целых чисел. Исключить из списка все нулевые элементы, задавая порядок чисел массивом индексов следующих элементов (next). В результате массив чисел остается без изменений, массив индексов переупорядочивается. Найти сумму все четных элементов списка. Результирующий список и сумму вывести на экран.

8. Создать односвязный список с помощью массива целых чисел. Ввести с клавиатуры число и поместить его за пятым элементом списка, задавая порядок чисел массивом индексов следующих элементов (next). Результирующий список вывести на экран.

9. Создать односвязный список с помощью массива целых чисел. Ввести с клавиатуры число и поместить его перед тем элементом списка, который больше него. Результирующий список вывести на экран. Порядок чисел в списке задается массивом индексов следующих элементов (next).

10. Создать односвязный список с помощью массива целых чисел. Ввести с клавиатуры число, найти это число в списке и удалить. Результирующий список вывести на экран. Порядок чисел в списке задается массивом индексов следующих элементов (next).

11. Создать односвязный список с помощью массива целых чисел. Ввести с клавиатуры число, найти все элементы с этим числом в списке и удалить. Результирующий список вывести на экран. Порядок чисел в списке задается массивом индексов следующих элементов (next).

12. Создать односвязный список с помощью массива целых чисел. Поменять местами четные и нечетные элементы списка (рядом стоящие). Результирующий список вывести на экран. Порядок чисел в списке задается массивом индексов следующих элементов (next).

Библиотека БГУИР

13. Создать односвязный список с помощью массива целых чисел. Сформировать новый список, в котором элементы следуют от конца к началу (последний элемент станет первым, предпоследний – вторым и т.д.). Результирующий список вывести на экран. Порядок чисел в списке задается массивом индексов следующих элементов (next).

14. Создать односвязный список с помощью массива целых чисел. Продублировать в списке первый, третий и пятый элементы. Результирующий список вывести на экран. Порядок чисел в списке задается массивом индексов следующих элементов (next).

15. Создать односвязный список с помощью массива целых чисел. Удалить в списке первый, третий и пятый элементы. Результирующий список вывести на экран. Порядок чисел в списке задается массивом индексов следующих элементов (next).

16. Создать список с помощью массива структур. Элементы списка в обратном порядке вывести на экран.

17. Создать список с помощью массива структур. Все четные элементы списка вывести на экран.

18. Создать список с помощью массива структур. Все нечетные элементы списка вывести на экран.

19. Создать односвязный список с помощью массива структур. Отсортировать элементы списка по возрастанию. Результирующий список вывести на экран.

20. Создать односвязный список с помощью массива структур. Один из элементов структуры – целое число. Расположить в начале списка все элементы списка с четными целыми числами. Результирующий список вывести на экран.

21. Создать односвязный список с помощью массива структур. Исключить из списка все элементы с нулевым целым числом. Результирующий список вывести на экран.

22. Создать односвязный список с помощью массива структур. Исключить из списка все элементы с нулевым целым числом. Найти сумму тех целочисленных элементов списка, которые являются четными. Результирующий список и сумму вывести на экран.

23. Создать односвязный список с помощью массива структур. Создать новый элемент списка, ввести с клавиатуры число и поместить его в целочисленное поле нового элемента. Разместить новый элемент за пятым элементом списка. Результирующий список вывести на экран.

24. Создать односвязный список с помощью массива структур. Создать новый элемент списка, ввести с клавиатуры число и поместить его в целочисленное поле нового элемента. Разместить новый элемент перед тем

Библиотека БГУИР

элементом списка, целочисленное поле которого имеет большее значение. Результирующий список вывести на экран.

25. Создать односвязный список с помощью массива структур. Ввести с клавиатуры число, найти это число в списке (в целочисленном поле) и удалить соответствующий элемент списка. Результирующий список вывести на экран.

26. Создать односвязный список с помощью массива структур. Ввести с клавиатуры число, найти все элементы с этим числом (в целочисленном поле) в списке и удалить. Результирующий список вывести на экран.

27. Создать односвязный список с помощью массива структур. Поменять местами четные и нечетные элементы списка (рядом стоящие). Результирующий список вывести на экран.

28. Создать односвязный список с помощью массива структур. Сформировать новый список, в котором элементы следуют от конца к началу (последний элемент станет первым, предпоследний – вторым и т.д.). Результирующий список вывести на экран.

29. Создать односвязный список с помощью массива структур. Продублировать в списке первый, третий и пятый элементы. Результирующий список вывести на экран.

30. Создать односвязный список с помощью массива структур. Удалить в списке первый, третий и пятый элементы. Результирующий список вывести на экран.

31. Написать программу, содержащую процедуру, которая меняет местами первый и второй элементы непустого списка. Если элементы не найдены, то выдать на экран соответствующее сообщение.

32. Написать программу, содержащую процедуру, которая меняет местами первый и пятый элементы непустого списка. Если элементы не найдены, то выдать на экран соответствующие сообщение.

33. Написать программу, содержащую процедуру, которая меняет местами первый и последний элементы непустого списка. Если элементы не найдены, то выдать на экран соответствующие сообщение.

34. Написать программу, содержащую процедуру, которая вставляет новый элемент перед каждым вхождением заданного элемента. Если элементы не найдены, то выдать на экран соответствующие сообщение.

35. Написать программу, содержащую процедуру, которая вставляет новый элемент за каждым вхождением заданного элемента. Если элементы не найдены, то выдать на экран соответствующие сообщение.

36. Написать программу, содержащую подпрограмму, которая проверяет на равенство списки M1 и M2.

Библиотека БГУИР

37. Написать программу, содержащую функцию, которая определяет, входит ли список M1 в список M2. Предполагается, что списки существуют.

38. Написать программу, содержащую подпрограмму, которая копирует в конец непустого списка M его первый элемент. Если элементы не найдены, то выдать на экран соответствующие сообщение.

39. Написать программу, содержащую подпрограмму, которая копирует в начало непустого списка M его последний элемент. Если элементы не найдены, то выдать на экран соответствующие сообщение.

40. Написать программу, содержащую процедуру, которая копирует в список M за каждым вхождением заданного элемента все элементы списка M1.

41. Написать программу, содержащую процедуру, которая объединяет два упорядоченных по неубыванию списка M1 и M2 в один упорядоченный по неубыванию список, построив новый список M.

42. Написать программу, содержащую процедуру, которая объединяет два упорядоченных по неубыванию списка M1 и M2 в один упорядоченный по неубыванию список, сменив соответствующим образом ссылки в M1 и M2.

43. Написать программу, содержащую функцию, которая проверяет, упорядочены ли элементы списка по алфавиту.

44. Написать программу сортировки существующего списка по алфавиту. В программе использовать подпрограммы.

45. Написать программу, которая создавала бы файл целых чисел, а затем формировала список целых чисел файла. Создать в конце списка элемент, содержащий сумму всех чисел файла. В программе использовать подпрограммы.

46. Написать программу, которая создавала бы файл целых чисел, а затем формировала список целых чисел файла. Создать список чисел, являющихся суммой соседних элементов. В программе использовать подпрограммы.

47. Написать программу, которая создавала бы текстовый файл, а затем формировала список строк файла. Создать список обратных строк. В программе использовать подпрограммы.

48. Написать программу, которая создавала бы текстовый файл, а затем формировала список строк файла. Создать отсортированный список строк. В программе использовать подпрограммы.

49. Написать программу, которая создавала бы файл комбинированного типа, а затем формировала список, используя какое-либо поле записи. Создать отсортированный список. В программе использовать подпрограммы.

50. Написать программу, которая создавала бы файл комбинированного типа, а затем формировала список элементов файла. Создать отсортированный по какому-либо полю список. В программе использовать подпрограммы.

Библиотека БГУИР

51. Составить программу, которая вводит с клавиатуры названия городов, динамически отводит место в памяти под каждое название и строит из них связанный список, упорядоченный по алфавиту. По окончании формирования список городов вывести на экран монитора.

52. Составить список учебной группы, содержащий не менее 15 студентов. Указать для каждого студента оценки, полученные на последних четырех экзаменах. Разработать программу, которая вводит данные с клавиатуры о каждом студенте, строит односвязный список, а затем удаляет из списка элементы, относящиеся к неуспевающим студентам.

53. Информацию о величине экспорта и соответствующий номер контракта записать в двусвязный кольцевой список. Затем переместить данную информацию в двумерный динамический массив и осуществить поиск максимальной величины экспорта. На экран вывести искомую информацию.

54. Создать двусвязный список, содержащий следующую информацию: год и соответствующую численность населения. Программу организовать таким образом, чтобы на экран выводилась информация, численность населения в которой была больше введенного с клавиатуры значения.

55. Ввести с клавиатуры строку символов, формируя из ее элементов двусвязный список. Написать программу, которая формирует двусвязный список из входной строки. В поле данных каждого элемента списка записывается отдельный символ. В программе производится анализ первого символа входной строки: если это буква ***'A'***, то в конец списка добавляется еще одна буква ***'A'***, иначе из списка исключаются все буквы ***'A'***. Полученный результат выводится на экран.

ЛАБОРАТОРНАЯ РАБОТА №13

ОЧЕРЕДИ. ОПЕРАЦИИ НАД ОЧЕРЕДЯМИ. ДЕКИ

Цель: Ознакомиться с понятием «очередь». Ознакомиться с операциями над очередями. Ознакомиться с понятием «дек». Изучить правила программной реализации очереди на основе статического массива.

Теоретические сведения

1. Понятие очереди. Операции над очередями. Кольцевая очередь. Дек

Очередью FIFO (First – In – First – Out – «первым пришел – первым исключается») называется такой последовательный список переменной длины, в котором включение элементов выполняется только с одной стороны списка (эту сторону часто называют концом или хвостом очереди), а исключение – с

Библиотека БГУИР

другой (называемой началом или головой очереди). Очереди к прилавкам и к кассам являются типичным бытовым примером очереди FIFO.

Основные операции над очередью – те же, что и над стеком, – включение, исключение, определение размера, очистка, неразрушающее чтение.

При представлении очереди массивом в дополнение к нему необходимы параметры-указатели: на начало очереди (на первый элемент в очереди) и на ее конец (первый свободный элемент в очереди). При включении элемента в очередь элемент записывается по адресу, определяемому указателем на конец, после чего этот указатель увеличивается на единицу. При исключении элемента из очереди выбирается элемент, адресуемый указателем на начало, после чего этот указатель также увеличивается на единицу (рис. 13.1).

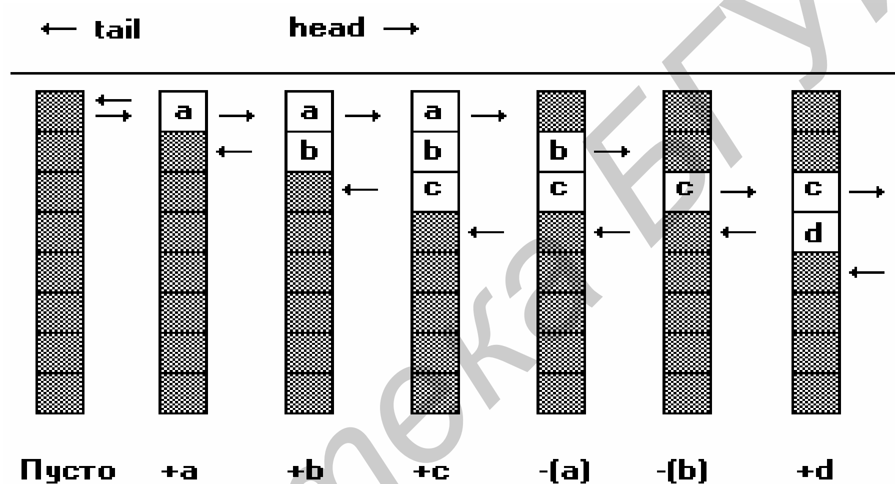


Рис. 13.1. Представление очереди массивом

Очевидно, что со временем указатель на конец при очередном включении элемента достигнет верхней границы той области памяти, которая выделена для очереди. Однако если операции включения чередовались с операциями исключения элементов, то в начальной части отведенной под очередь памяти имеется свободное место. Для того чтобы места, занимаемые исключенными элементами, могли быть повторно использованы, очередь замыкается в кольцо: указатели (на начало и на конец), достигнув конца выделенной области памяти, переключаются на ее начало. Такая организация очереди в памяти называется **кольцевой очередью**.

Возможны, конечно, и другие варианты организации: например, всякий раз, когда указатель конца достигнет верхней границы памяти, сдвигать все непустые элементы очереди к началу области памяти, но как этот, так и другие варианты требуют перемещения в памяти элементов очереди и менее эффективны, чем кольцевая очередь.

Библиотека БГУИР

В исходном состоянии указатели на начало и на конец указывают на начало области памяти. Равенство этих двух указателей (при любом их значении) является признаком пустой очереди. Если в процессе работы с кольцевой очередью число операций включения превышает число операций исключения, то может возникнуть ситуация, в которой указатель конца «догонит» указатель начала. Это ситуация заполненной очереди, но если в этой ситуации указатели сравняются, эта ситуация будет неотличима от ситуации пустой очереди. Для различения этих двух ситуаций к кольцевой очереди предъявляется требование, чтобы между указателем конца и указателем начала оставался «зазор» из свободных элементов. Когда этот «зазор» сокращается до одного элемента, очередь считается заполненной и дальнейшие попытки записи в нее блокируются. Очистка очереди сводится к записи одного и того же (в общем случае не обязательно начального) значения в оба указателя. Определение размера состоит в вычислении разности указателей с учетом кольцевой природы очереди.

Дек – особый вид очереди. Дек (от англ. deq – double ended queue, т.е. очередь с двумя концами) – это такой последовательный список, в котором как включение, так и исключение элементов может осуществляться с любого из двух концов списка. Частный случай дека – дек с ограниченным входом и дек с ограниченным выходом. Логическая и физическая структуры дека аналогичны логической и физической структуре кольцевой FIFO-очереди. Однако применительно к деку целесообразно говорить не о начале и конце, а о левом и правом конце.

Операции над деками:

- включение элемента справа;
- включение элемента слева;
- исключение элемента справа;
- исключение элемента слева;
- определение размера;
- очистка.

Физическая структура дека в статической памяти идентична структуре кольцевой очереди. Динамическая реализация является очевидным объединением стека и очереди.

Задачи, требующие структуры дека, встречаются в вычислительной технике и программировании гораздо реже, чем задачи, реализуемые на структуре стека или очереди. Как правило, вся организация дека выполняется программистом без каких-либо специальных средств системной поддержки.

Библиотека БГУИР

Примером дека может быть, например, некий терминал, в который вводятся команды, каждая из которых выполняется какое-то время. Если ввести следующую команду, не дождавшись, пока закончится выполнение предыдущей, то она встанет в очередь и начнет выполняться, как только освободится терминал. Это FIFO-очередь. Если же ввести дополнительно операцию отмены последней введенной команды, то получается дек.

2. Программная реализация очереди на основе массива

Пример 13.1

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <malloc.h>

////////////// Queue for int positive (x>0)

typedef struct
{
  int *queue;
  int head;
  int tail;
  int size;
} QUEUE;

int Init(QUEUE* Queue, int nSize);
int Extract(QUEUE* Queue);
int Add(QUEUE* Queue, int nElement);
void Clear(QUEUE* Queue);
int GetSize(QUEUE* Queue);
void Free(QUEUE* Queue);
```

Программа реализации кольцевой очереди на основе массива – queue.cpp:

```
#include "queue.h"
////////////// Queue for int positive (x>0)
////////////////////////////////////////
int Init(QUEUE* Queue, int nSize)

{
  Queue->queue = (int*)malloc(nSize*(sizeof(int)));
```


Библиотека БГУИР

```
  if (Queue->queue == NULL) return -1;
  Queue->size = nSize;
  Queue->head = 0;
  Queue->tail = 0;
  return nSize;
}

//////////////////////////////////////////
int Extract(QUEUE* Queue)
{
  int w;
  if (Queue->head == Queue->tail) return -1;
  w = Queue->queue[Queue->head];
    if (Queue->head == Queue->size - 1)
    Queue->head = 0;
  else
    Queue->head++;
  return w;
}
//////////////////////////////////////////
int Add(QUEUE* Queue, int nElement)
{
  if (Queue->tail +1 == Queue->head)
    return -1;      // Очередь заполнена - запись невозможна!

  if (Queue->tail == Queue->size )
  Queue->tail=0;  //Достигнут конец памяти, выделенной под очередь
  Queue->queue[Queue->tail++] = nElement;
 return 1;
}
//////////////////////////////////////////
void Clear(QUEUE* Queue)
{
  Queue->head = 0;
  Queue->tail = 0;
}
//////////////////////////////////////////
int GetSize(QUEUE* Queue)
{
```

Библиотека БГУИР

```
  if (Queue->head == Queue->tail)
    return 0;      // Очередь пуста
  if (Queue->head < Queue->tail)
    return (Queue->tail - Queue->head);
  else
    return (Queue->size - (Queue->head - Queue->tail));
}
//////////////////////////////////////////
void Free(QUEUE* Queue)
{   free(Queue->queue);  }
```

Тестовая программа:

```
#include "queue.h"

////////////////////////// Print of queue information
void Print(QUEUE *Queue)
{
if (Queue->head == Queue->tail)
      printf("\nQueue is empty!");
else
printf ("\nhead=%u, tail=%u, (%d-%d)", Queue->head, Queue->tail,
  Queue->queue[Queue->head], Queue->queue[Queue->tail-1]);
}
void main()
{
QUEUE Queue;
Init(&Queue,4);  Print(&Queue);
Add(&Queue,10);  Print(&Queue);
Add(&Queue,11);  Print(&Queue);
Add(&Queue,12);  Print(&Queue);
Extract(&Queue); Print(&Queue);
Add(&Queue,13);  Print(&Queue);
Extract(&Queue); Print(&Queue);
Add(&Queue,21);  Print(&Queue);
Extract(&Queue); Print(&Queue);
Add(&Queue,22);  Print(&Queue);
GetSize(&Queue);
Extract(&Queue); Print(&Queue);
Extract(&Queue); Print(&Queue);
Extract(&Queue); Print(&Queue);
```

Библиотека БГУИР

```
Clear(&Queue);   Print(&Queue);
Add(&Queue,31);  Print(&Queue);
Extract(&Queue); Print(&Queue);
Free(&Queue);
getch();
}
```

Варианты индивидуальных заданий

1. Создать очередь для целых (положительных и отрицательных) чисел. Максимальный размер очереди вводится с экрана. Создать функции для ввода и вывода элементов очереди. Ввести 6 элементов (положительных и отрицательных). Вывести 2 первых отрицательных элемента очереди.

2. Создать очередь для массива целых (положительных и отрицательных) чисел. Максимальный размер очереди вводится с экрана. Создать функции для ввода и вывода элементов очереди. Ввести 6 элементов. При вводе чисел в очередь попадают только отрицательные элементы. Вывести все элементы очереди.

3. Создать очередь для целых (положительных и отрицательных) чисел. Максимальный размер очереди вводится с экрана. Создать функции для ввода, вывода и определения размера очереди. Ввести 6 элементов. Вывести 2 первых элемента очереди. Вывести размер оставшейся очереди.

4. Создать очередь для целых (положительных и отрицательных) чисел. Максимальный размер очереди вводится с экрана. Создать функции для ввода, вывода и определения размера очереди. Ввести 6 элементов. Вывести элементы очереди до первого отрицательного (включительно). Вывести размер оставшейся очереди.

5. Создать очередь для символов. Максимальный размер очереди вводится с экрана. Создать функции для ввода и вывода элементов очереди. Ввести эталонный символ. Вводить символы с экрана в очередь до встречи эталонного. Вывести все элементы очереди.

6. Создать очередь для символов. Максимальный размер очереди вводится с экрана. Создать функции для ввода и вывода элементов очереди. Вводить символы с экрана в очередь. После введения 3-го символа в ответ на каждый вводимый выводить крайний левый символ.

7. Создать очередь для символов. Максимальный размер очереди вводится с экрана. Создать функции для ввода, вывода и определения размера очереди. Ввести эталонный символ. Вводить символы с экрана в очередь до встречи эталонного. Вывести размер очереди.

Библиотека БГУИР

8. Создать очередь для символов. Максимальный размер очереди вводится с экрана. Создать функции для ввода и вывода элементов очереди. Вводить символы с экрана в очередь. В случае совпадения вводимого символа с последним элементом очереди выводить первый элемент очереди.

9. Создать очередь для символов. Максимальный размер очереди вводится с экрана. Создать функции для ввода, вывода и определения размера очереди. Вводить символы с экрана в очередь. В случае совпадения вводимого символа с последним элементом очереди выводить размер очереди.

10. Создать очередь для символов. Максимальный размер очереди вводится с экрана. Создать функции для ввода и вывода элементов очереди. Добавлять символы с экрана в очередь. В случае совпадения вводимого символа с последним элементом очереди удалять и выводить на экран все элементы очереди.

11. Создать очередь для символов. Максимальный размер очереди вводится с экрана. Создать функции для ввода и вывода элементов очереди. Вводить символы с экрана. В случае совпадения вводимого символа с последним элементом очереди в очередь его не добавлять, а удалять первый элемент.

12. Создать очередь для символов. Максимальный размер очереди вводится с экрана. Создать функции для ввода и вывода элементов очереди. Вводить символы с экрана. В случае совпадения вводимого символа с последним элементом очереди выводить размер очереди.

13. Создать очередь для целых чисел. Максимальный размер очереди вводится с экрана. Создать функции для ввода и вывода элементов очереди. Ввести в очередь числа с экрана. После этого перейти в режим, при котором при каждом вводе числа из очереди удаляется первый элемент, и если он совпадает с введенным числом, то он добавляется в очередь.

14. Создать очередь для целых чисел. Максимальный размер очереди вводится с экрана. Создать функции для ввода и вывода элементов очереди. Ввести в очередь числа с экрана. После этого перейти в режим, при котором при каждом вводе числа выводится первый элемент очереди, и если он не совпадает с введенным числом, то оно заносится в очередь.

15. Создать очередь для целых чисел. Максимальный размер очереди вводится с экрана. Создать функции для ввода и вывода элементов очереди. Ввести в очередь числа с экрана. После этого перейти в режим ввода, при котором перед добавлением элемента происходит удаление одного элемента.

16. Создать дек для целых (положительных и отрицательных) чисел. Максимальный размер дека вводится с экрана. Создать функции для ввода и

Библиотека БГУИР

вывода элементов дека. Ввести 6 элементов (положительных и отрицательных). Вывести 2 первых правых отрицательных элемента дека.

17. Создать дек для массива целых чисел. Максимальный размер дека вводится с экрана. Создать функции для ввода и вывода элементов дека. Ввести 3 элемента справа и 3 слева. При вводе чисел в дек попадают только отрицательные элементы. Вывести все элементы дека.

18. Создать дек для целых (положительных и отрицательных) чисел. Максимальный размер дека вводится с экрана. Создать функции для ввода, вывода и определения размера дека. Ввести 3 элемента справа и 3 слева. Вывести по одному элементу справа и слева. Вывести размер оставшегося дека.

19. Создать дек для целых (положительных и отрицательных) чисел. Максимальный размер дека вводится с экрана. Создать функции для ввода, вывода и определения размера дека. Ввести 3 элемента справа и 3 слева. Вывести элементы дека справа до первого отрицательного (включительно). Вывести размер оставшегося дека.

20. Создать дек для символов. Максимальный размер дека вводится с экрана. Создать функции для ввода и вывода элементов дека. Ввести эталонный символ. Вводить символы в дек поочередно справа и слева до встречи эталонного. Вывести все элементы дека.

21. Создать дек для символов. Максимальный размер дека вводится с экрана. Создать функции для ввода и вывода элементов дека. Вводить элементы в дек поочередно справа и слева. При этом в ответ на добавление элемента с противоположной стороны дека один элемент удаляется.

22. Создать дек для символов. Максимальный размер дека вводится с экрана. Создать функции для ввода, вывода и определения размера дека. Ввести эталонный символ. Вводить символы в дек поочередно справа и слева до встречи эталонного. Вывести размер дека.

23. Создать дек для символов. Максимальный размер дека вводится с экрана. Создать функции для ввода и вывода элементов дека. Добавлять символы в дек поочередно справа и слева. В случае совпадения добавляемого символа с элементом на другом конце дека выводить его на экран.

24. Создать дек для символов. Максимальный размер дека вводится с экрана. Создать функции для ввода, вывода и определения размера дека. Добавлять символы в дек поочередно справа и слева. В случае совпадения добавляемого символа с элементом на другом конце дека выводить размер дека.

25. Создать дек для символов. Максимальный размер дека вводится с экрана. Создать функции для ввода и вывода элементов дека. Добавлять символы в дек поочередно справа и слева. В случае совпадения добавляемого

Библиотека БГУИР

символа с концом дека вывести на экран все элементы со стороны совпавшего.

26. Создать дек для символов. Максимальный размер дека вводится с экрана. Создать функции для ввода и вывода элементов дека. Добавлять символы в дек поочередно справа и слева. В случае совпадения введенного символа с элементом соответствующего конца дека его не добавлять.

27. Создать дек для символов. Максимальный размер дека вводится с экрана. Создать функции для ввода, вывода и определения размера дека. Ввести в дек символы с экрана. Вводить символы с экрана. В случае совпадения вводимого символа с одним из концов дека выводить его размер.

28. Создать дек для целых чисел. Максимальный размер дека вводится с экрана. Создать функции для ввода и вывода элементов дека. При каждом вводе числа слева удаляется элемент, и если он не совпадает с введенным числом, то введенное число добавляется справа.

29. Создать дек для целых чисел. Максимальный размер дека вводится с экрана. Создать функции для ввода и вывода элементов дека. Ввести в дек числа с экрана. После этого перейти в режим, при котором слева удаляется элемент, и если он совпадает с введенным числом, то введенное число добавляется справа, а иначе – слева.

30. Создать дек для плавающих чисел. Максимальный размер дека вводится с экрана. Создать функции для ввода и вывода элементов дека. Ввести в дек числа с экрана. После этого перейти в режим ввода, при котором перед занесением элемента происходит удаление левого элемента.

31. Дан текстовый файл. Проанализировав в программе содержимое файла, выбрать из него числа и занести в очередь. Вывести содержимое очереди на экран и посчитать сумму этих чисел. Решение в программе оформить через подпрограммы.

32. Дан текстовый файл. Проанализировав в программе содержимое файла, выбрать из него имена и занести в очередь. Вывести содержимое очереди на экран и посчитать количество элементов образованной очереди. Решение в программе оформить через подпрограммы.

33. Проверить на равенство две очереди. Решение в программе оформить через подпрограммы.

34. Найти среди 3-х (4, 5) очередей две одинаковые. Решение в программе оформить через подпрограммы.

35. Организовать три очереди с одинаковым количеством элементов, содержащие соответственно имена, отчества и фамилии людей. Составить очередь из элементов, содержащих наиболее полную информацию о людях,

Библиотека БГУИР

воспользовавшись уже созданными очередями и запросив какую-то дополнительную информацию. Решение в программе оформить через подпрограммы.

36. Создать файл символьного типа. Организовывая очереди по N элементов, создать файл слов по N символов в каждом. Решение в программе оформить через подпрограммы.

37. Создать файл целого типа. Проанализировав в программе содержимое файла, создать одну очередь однозначных чисел, а вторую – двузначных. Перемножить соответственные элементы двух очередей и организовать третью очередь. Результат вывести в текстовый файл. Решение в программе оформить через подпрограммы.

38. Используя очередь, проверить, какие строки текстового файла являются симметричными. Решение в программе оформить через подпрограммы.

39. Используя очередь, проверить на равенство два текстовых файла. Решение в программе оформить через подпрограммы.

40. Создать две очереди. Проверить, является ли одна из очередей частью другой. Решение в программе оформить через подпрограммы.

ЛАБОРАТОРНАЯ РАБОТА №14

СТЕКИ. ОЧЕРЕДИ. ОПЕРАЦИИ НАД СТЕКАМИ И ОЧЕРЕДЯМИ

Цель: Ознакомиться с понятиями «стек» и «очередь». Ознакомиться с операциями над стеками и очередями. Изучить правила программной реализация стека и очереди на основе статического массива.

Теоретические сведения

1. Понятие стека. Операции над стеком

Стек – это последовательный список переменной длины, включение и исключение элементов из которого выполняются только с одной стороны списка, называемого **вершиной** стека. Другие названия стека – «магазин» и «очередь, функционирующая по принципу LIFO» (Last – In – First – Out – «последним пришел – первым исключается»). Примеры стека: винтовочный патронный магазин, тупиковый железнодорожный разъезд для сортировки вагонов.

Основные операции над стеком – включение нового элемента (англ. push – заталкивать) и исключение элемента из стека (англ. pop – выскакивать).

Библиотека БГУИР

Полезными могут быть также вспомогательные операции:

- определение текущего числа элементов в стеке;
- очистка стека;
- неразрушающее чтение элемента из вершины стека, которое может быть реализовано как комбинация основных операций:

x = pop(stack0);   push(stack0,x).

Для наглядности рассмотрим небольшой пример, демонстрирующий принцип включения элементов в стек и исключения элементов из стека. На рис. 14.1 изображены следующие состояния стека:

а) пустого;

б–г) после последовательного включения в него элементов с именами 'A', 'B', 'C';

д, е) после последовательного удаления из стека элементов 'C' и 'B';

ж) после включения в стек элемента 'D'.

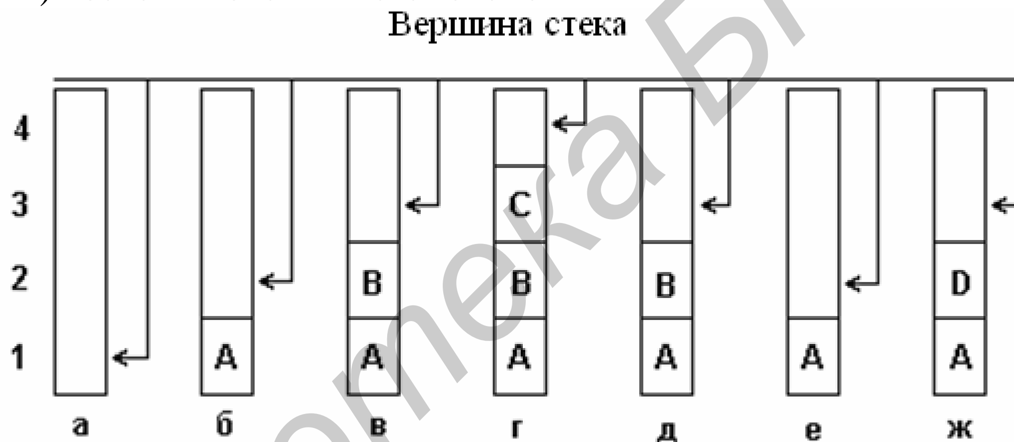


Рис. 14.1. Включение и исключение элементов из стека

Стек можно представить, например, в виде стопки книг (элементов), лежащей на столе. Присвоим каждой книге свое название, например A, B, C, D... Тогда в момент времени, когда на столе книги отсутствуют, про стек по аналогии можно сказать, что он пуст, т.е. не содержит ни одного элемента. Если же начинать последовательно класть книги одну на другую, то получим стопку книг (допустим из n книг), или получим стек, в котором содержится n элементов, причем вершиной его будет являться элемент n + 1. Удаление элементов из стека осуществляется аналогичным образом, т.е. удаляется последовательно по одному элементу, начиная с вершины, или по одной книге из стопки.

Реализация стека может быть выполнена на основе массива. Кроме собственно массива необходимо дополнительно иметь переменную (назовем ее «указатель стека»), адресующую вершину стека. Под вершиной стека можно

Библиотека БГУИР

понимать либо первый свободный элемент стека, либо последний записанный. Все равно, какой из этих двух вариантов выбрать, важно впоследствии при обработке стека строго его придерживаться.

При занесении элемента в стек элемент записывается на место, определяемое указателем стека, затем этот указатель модифицируется таким образом, чтобы он указывал на следующий свободный элемент.

Операция выталкивания элемента состоит в модификации указателя стека (в направлении обратном модификации при включении) и выборке значения, на которое указывает указатель стека. После выборки элемент, в котором размещалось выбранное значение, считается свободным.

2. Программная реализация стека на основе массива

Пример 14.1

Файл stack.h

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <conio.h>
//////////////////// Stack for char
typedef struct
{char *stack;
  int head;
  int size;
} STACK;
int Init(STACK* Stack, int nSize);
char Pop(STACK* Stack);
int Push(STACK* Stack, char cElement);
void Clear(STACK* Stack);
int GetSize(STACK* Stack);
void Free(STACK* Stack);

#include "stack.h"
//////////////////// Stack for char
//////////////////////////////////////////////
int Init(STACK* Stack, int nSize)
{Stack->stack = (char*)malloc(nSize);
if (Stack->stack == NULL) return -1;
Stack->size = nSize;
Stack->head = 0;
```

Библиотека БГУИР

```
return nSize;
}
//////////////////////////////////////////////
char Pop(STACK* Stack)
{if (Stack->head == 0) return 0;
 return Stack->stack[--Stack->head]; }
//////////////////////////////////////////////
int Push(STACK* Stack, char cElement)
{if (Stack->head == Stack->size) return -1;
 Stack->stack[Stack->head++] = cElement;
 return 1;
}
//////////////////////////////////////////////
void Clear(STACK* Stack)
{Stack->head = 0;}
//////////////////////////////////////////////
int GetSize(STACK* Stack)
{  return Stack->head;
}
//////////////////////////////////////////////
void Free(STACK* Stack)
{ free(Stack->stack);  }

void main()
{STACK A;
 Init(&A, 8);
 Push(&A,  'J');
 Push(&A,  'F');
 char c = Pop(&A);
```

. . .

Варианты индивидуальных заданий

1. Создать стек для целых чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Добавить 6 элементов. Удалить и вывести на экран 2 элемента.

2. Создать стек для целых (положительных и отрицательных) чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Ввести с экрана 6 элементов. При вводе чисел в стек попадают только отрицательные элементы. Вывести все элементы стека.

Библиотека БГУИР

3. Создать стек для целых чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Ввести с экрана 6 элементов. Удалить 2 элемента. Вывести размер стека.

4. Создать стек для целых (положительных и отрицательных) чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Вводить с экрана числа, причем в стек должны добавляться поочередно положительные и отрицательные числа.

5. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Ввести эталонный символ. Вводить символы с экрана в стек до встречи эталонного. Вывести все элементы стека.

6. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Добавить символы с экрана в стек. После добавления 5-го символа перед добавлением удалять элемент из стека.

7. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Ввести эталонный символ. Вводить символы с экрана в стек до встречи эталонного. Вывести размер стека.

8. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Добавлять символы с экрана в стек. В случае совпадения вводимого символа с вершиной стека вытолкнуть его и распечатать ее.

9. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Вводить символы с экрана в стек. В случае совпадения вводимого символа с вершиной стека вывести размер стека.

10. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана в первый стек. В случае совпадения вводимого символа с вершиной стека вводить во второй стек.

11. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана в стеки поочередно.

12. Создать стек для символов и стек для чисел. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Символ попадает в первый стек, а его численное представление – во второй.

Библиотека БГУИР

13. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Прописные буквы попадают в первый стек, строчные – во второй, остальные символы пропускаются.

14. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Прописные буквы преобразуются в строчные и попадают в первый стек, строчные преобразуются в прописные и попадают во второй, остальные символы пропускаются.

15. Создать стек для целых чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Числовое представление символа попадает в стек.

16. Создать стек для целых чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Добавить 6 элементов. Удалить и вывести на экран 2 элемента. Задачу решить с использованием механизма указателей.

17. Создать стек для целых (положительных и отрицательных) чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Ввести 6 элементов. При вводе чисел в стек попадают только отрицательные элементы. Вывести все элементы стека. Задачу решить с использованием механизма указателей.

18. Создать стек для целых чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Ввести с экрана 6 элементов. Удалить 2 элемента. Вывести размер стека. Задачу решить с использованием механизма указателей.

19. Создать стек для целых (положительных и отрицательных) чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Вводить с экрана числа, причем в стек должны добавляться поочередно положительные и отрицательные числа. Задачу решить с использованием механизма указателей.

20. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Ввести эталонный символ. Вводить символы с экрана в стек до встречи эталонного. Вывести все элементы стека. Задачу решить с использованием механизма указателей.

21. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Добавить символы с экрана в стек. После добавления 5-го символа перед добавлением

Библиотека БГУИР

следующего удалять элемент из стека. Задачу решить с использованием механизма указателей.

22. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Ввести эталонный символ. Вводить символы с экрана в стек до встречи эталонного. Вывести размер стека. Задачу решить с использованием механизма указателей.

23. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Добавлять символы с экрана в стек. В случае совпадения вводимого символа с вершиной стека вытолкнуть его и распечатать ее. Задачу решить с использованием механизма указателей.

24. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Вводить символы с экрана в стек. В случае совпадения вводимого символа с вершиной стека вывести размер стека. Задачу решить с использованием механизма указателей.

25. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана в первый стек. В случае совпадения вводимого символа с вершиной стека вводить во второй стек. Задачу решить с использованием механизма указателей.

26. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана в стеки поочередно. Задачу решить с использованием механизма указателей.

27. Создать стек для символов и стек для чисел. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Символ попадает в первый стек, а его численное представление – во второй. Задачу решить с использованием механизма указателей.

28. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Прописные буквы попадают в первый стек, строчные – во второй, остальные символы пропускаются. Задачу решить с использованием механизма указателей.

29. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Прописные буквы преобразуются в строчные и попадают в

Библиотека БГУИР

первый стек, строчные преобразуются в прописные и попадают во второй, остальные символы пропускаются. Задачу решить с использованием механизма указателей.

30. Создать стек для целых чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Числовое представление символа попадает в стек. Задачу решить с использованием механизма указателей.

31. Создать текстовый файл, содержащий текстовую и числовую информацию. Используя стек, создать другой текстовый файл, в котором числа были бы записаны в обратном порядке.

32. Создать текстовый файл, содержащий текстовую информацию. Используя стек, создать другой текстовый файл, в котором слова были бы записаны в обратном порядке.

33. Создать текстовый файл, содержащий некоторую информацию. Используя стек, создать другой текстовый файл, в котором строки были бы записаны в обратном порядке.

34. Создать текстовые файлы, содержащие один текстовую, а другой числовую информацию (количество слов и чисел должно быть одинаковым). Используя стек, создать другой текстовый файл, в котором числа и слова чередовались бы и были бы записаны в обратном порядке.

35. Создать текстовые файлы, содержащие один текстовую, а другой числовую информацию (количество слов и чисел должно быть одинаковым). Используя стек, создать другой текстовый файл, в котором числа и слова чередовались бы, а порядок чисел и слов был бы сохранен.

36. Создать текстовые файлы, содержащие один текстовую, а другой числовую информацию (количество слов и чисел может быть неодинаковым). Используя стек, создать другой текстовый файл, в котором числа и слова чередовались бы и были бы записаны в обратном порядке («лишние» числа или слова были бы записаны в конец файла).

37. В файле находится текст программы на Паскале. Используя стек, проверить правильность вложений циклов в этой программе.

38. В файле находится текст программы на Паскале. Используя стек, проверить правильность вложений операторных скобок (begin – end) в этой программе.

39. В файле записан текст, сбалансированный по круглым скобкам. Требуется для каждой пары соответствующих открывающей и закрывающей скобок напечатать номера их позиций в тексте, упорядочив пары номеров по возрастанию номеров позиций закрывающих скобок. Например, для текста a + (45 – f(x)*(b – c)) надо напечатать 8 10, 12 16, 3 17.

Библиотека БГУИР

40. В текстовом файле без ошибок записано логическое выражение следующего вида: <лог.выр> :: = true | false | <лог.выр> and <лог.выр> | <лог.выр> or <лог.выр>. Используя стек, вычислить значение этого выражения с учетом общепринятого приоритета операций.

41. Написать программу слияния двух стеков, содержащих возрастающую последовательность целых положительных чисел, в третий стек так, чтобы его элементы располагались также в порядке возрастания.

42. Написать программу формирования стека, куда помещаются целые положительные числа, вводимые с клавиатуры (процесс ввода должен прекращаться, как только среди вводимых чисел появляется отрицательное число, после этого программа должна вывести на экран содержимое стека в том же порядке, в котором они были введены).

ЛАБОРАТОРНАЯ РАБОТА №15

АЛГОРИТМЫ МЕТОДОВ СОРТИРОВКИ И ПОИСКА

Цель: Изучить существующие алгоритмы сортировки списков (массивов) и поиска их элементов и разработать программу для реализации этих методов.

Теоретические сведения

При работе со списками очень часто возникает необходимость перестановки элементов списка в определенном порядке. Такая задача называется сортировкой списка и для ее решения существуют различные методы. Рассмотрим некоторые из них (рис. 15.1).

Рис. 15.1 Виды прямых методов сортировки

Библиотека БГУИР

Пример 15.1

Функция для перемены местами элементов:

```
void swap(int   *x,   int   *y)
{
  int t = *x; /*промежуточная переменная*/
  /* Перемена данных местами  */
  *x = *y;
  *y = t;
}
```

1. Пузырьковая сортировка (методом обмена)

Задача сортировки заключается в следующем: задан список целых чисел (простейший случай) B=< K1, K2,..., Kn >. Требуется переставить элементы списка B так, чтобы получить упорядоченный список B'=< K'1, K'2,...,K'n >, в котором для любого 1<=i<=n элемент K'(i) <= K'(i + 1).

При обменной сортировке упорядоченный список B' получается из B систематическим обменом пары рядом стоящих элементов, не отвечающих требуемому порядку, пока такие пары существуют.

Наиболее простой метод систематического обмена соседних элементов с неправильным порядком при просмотре всего списка слева направо определяет пузырьковую сортировку: максимальные элементы как бы всплывают в конце списка.

Пример 15.2

Функция BubbleSort() реализует алгоритм сортировки методом «пузырька»:

```
void BubbleSort(int a[],int n)
{
      /*функции передается массив  и его размерность */
      int i, j;  /* переменные цикла */
      for (i=0; i<n; i++)
      for (j=n-1; j>i; j--)
      if (a[j-1] > a[j])
      /*если элемент "тяжелее" следующего
      swap(&a[j-1],&a[j]) /*поменять их местами */
}
```

Библиотека БГУИР

Анализ пузырьковой сортировки.

Пузырьковая сортировка обладает несколькими характеристиками:

- после каждой итерации только один элемент данных помещается в свою правильную позицию;
- сравнение и перестановка смежных элементов данных;
- в каждой итерации внутреннего цикла выполняется не более (n-iteration-1) перестановок;
- худший случай – когда элементы данных отсортированы в обратном порядке;
- лучший случай – когда элементы данных уже отсортированы в правильном порядке;
- легкость реализации.

2. Сортировка методом выбора

Алгоритм:

1 шаг – находится наименьший элемент в массиве;

2 шаг – найденный элемент меняется с первым элементом;

3 шаг – процесс повторяется с оставшимися n – 1 элементами, n – 2 элементами и так далее, пока в конце не останется самый большой элемент, который не нуждается в перестановке.

Пример 15.3

```
void MinSort(int a[], int n)
{  int i, j, k;
   for (i=0; i<n-1; i++)
{  for (k=i; j=i+1; j<n; j++) // находим в цикле
      if (a[j] < a[k])   // минимальный элемент
      {   k = j;         // запоминаем его номер в k
          swap(&a[k],&a[i]);// меняем местами минимальный и
      }           // элемент, с которого начинался цикл
}
}
```

3. Сортировка методом вставки

Сортировка вставками – очень простой метод сортировки, при котором элементы данных используются как ключи для сравнения. Этот алгоритм сначала упорядочивает A[0] и A[1], вставляя A[1] перед A[0], если A[0] > A[1]. Затем оставшиеся элементы данных по очереди вставляются в этот

Библиотека БГУИР

упорядоченный список. После k-й итерации элемент A[k] оказывается в своей правильной позиции и элементы от A[0] до A[k] уже отсортированы.

Пример 15.4

Функция InsertSort() реализует алгоритм сортировки методом вставки:

```
void InsertSort (int a[], int n)
{      int i, j
       for (i=1; i<=n-1; i++)
       {    j=i;
            while (a[j]<a[j-1] && j>=1)
             {   Swap(&a[j],&a[j-1]);
           j--;    }
        }
}
```

Анализ сортировки вставками

Сортировка вставками обладает следующими характеристиками:

• после каждой итерации только один элемент помещается в свою правильную позицию;

• при этом методе выполняется меньше перестановок, чем в пузырьковой сортировке;

• наихудший случай – когда все элементы данных отсортированы в обратном порядке;

• наилучший случай – когда элементы почти отсортированы в правильном порядке;

• легкость реализации.

4. Сортировка методом Шелла

Метод Шелла является обобщением метода вставок и основан на том, что сортировка методом вставок выполняется очень быстро на почти отсортированном массиве данных. Он также известен как сортировка с убывающим шагом. В отличие от сортировки методом вставок при сортировке методом Шелла весь массив не сортируется одновременно: массив разбивается на отдельные сегменты, которые сортируются раздельно с помощью метода вставок.

Основная идея алгоритма состоит в том, что на начальном этапе реализуется сравнение и, если требуется, перемещение далеко отстоящих друг от друга элементов. Интервал между сравниваемыми элементами (gap) постепенно уменьшается до единицы, что приводит к перестановке

Библиотека БГУИР

соседних элементов на последних стадиях сортировки (если это необходимо).

Реализуем метод Шелла следующим образом. Начальный шаг сортировки примем равным n/2, т.е. ½ от общей длины массива, и после каждого прохода будем уменьшать его в два раза. Каждый этап сортировки включает в себя проход всего массива и сравнение отстоящих на gap элементов. Проход с тем же шагом повторяется, если элементы переставлялись. Заметим, что после каждого этапа отстоящие на gap элементы отсортированы. Рассмотрим пример.

<u>Пример 15.5</u>

Рассортировать массив чисел: 41, 53, 11, 37, 79, 19, 7, 61. В строке после массива в круглых скобках указаны индексы сравниваемых элементов и указан номер внешнего цикла.

41 53 11 37 79 19 7 61 – исходный массив
42 19 11 37 79 19 7 61 – (0,4),(1,5)
41 19 7 37 79 19 11 61 – (2,6),(3,7) — 1-й цикл

7 19 41 37 11 53 79 61 – (0,2),(1,3),(2,4),(3,5),(4,6),(5,7)
7 19 11 37 41 53 79 61 – — 2-й цикл

7 11 19 37 41 53 61 79 – (сравнивались соседние элементы) 3-й цикл

```
void sort_shell(int *x,int n)
{   int i,j;     //две переменные цикла
    int gap;     //шаг сортировки
    int sorted;  //флаг окончания этапа сортировки
    for(gap=n/2;gap>0;gap/=2)//начало сортировки
     do
      {  sorted = 0;
        for(i=0,j=gap;j<n;i++,j++)
          if(*(x+i)>*(x+j))
           { swap((x+i),(x+j);
             sorted = 1;}
      }
    while(sorted);      }
```


Библиотека БГУИР

Анализ сортировки методом Шелла

Сортировка методом Шелла будет выполняться для многих задач, в которых количество элементов в массиве данных небольшое. Любую возможность даже незначительного ускорения сортировки необходимо использовать.

5. Сортировка методом Хоора

Сортировка методом Хоора (или быстрая сортировка) – это наиболее эффективный алгоритм внутренней сортировки. Его производительность зависит от выбора точки разбиения. При быстрой сортировке используется следующая стратегия:

1. Массив разбивается на меньшие подмассивы.
2. Подмассивы сортируются.
3. Отсортированные подмассивы объединяются.

Быструю сортировку можно реализовать несколькими способами, но цель каждого подхода заключается в выборе элемента данных и помещении его в правильную позицию (этот элемент называется точкой разбиения) таким образом, чтобы все элементы слева от точки разбиения оказались меньше (или предшествовали) точки разбиения, а все элементы справа от точки разбиения оказались больше (или следовали за ней). Выбор точки разбиения и метод, используемый для разбиения массива, оказывают большое влияние на общую производительность реализации.

Рассмотрим один из вариантов реализации сортировки Хоора.

Пример 15.6

В самой процедуре сортировки сначала выберем средний элемент. Потом, используя переменные i и j, пройдемся по массиву, отыскивая в левой части элементы больше среднего, а в правой – меньше среднего. Два найденных элемента переставим местами. Будем действовать так, пока i не станет больше j. Тогда получаем два подмножества, ограниченные с краев индексами l и r, а в середине – j и i. Если эти подмножества существуют (то есть $i < r$ и $j > l$), то выполним их сортировку.

```
void Quicksort(int a[], int n, int left, int right)
{          int i=left, j=right; /*Инициализируем переменные левой и
                                          правой границами подмассива*/
int test=a[(left+right)/2]; /*Выбираем в качестве
                  элемента разбиения средний элемент массива*/
do { while (a[i] < test)
```

```
        i++;
        /*находим элемент, больший элемента разбиения */
        while (a[j] > test)
        j--;
        /*находим элемент, меньший элемента разбиения */
        if (i<=j)
        {  Swap(&a[i], &a[j]);
           i++; j--;  }
         }
    while(i <= j);   /*рекурсивно вызываем алгоритм для
                       правого и левого подмассива*/
    if (i<right)
    QuickSort(a, n, i, right);
    if (j>left)
    QuickSort(a, n, left, j);
    }
```

Анализ быстрой сортировки

Быструю сортировку следует рассмотреть одной из первых при выборе метода внутренней сортировки. Алгоритм этой сортировки содержит сложную фазу разбиения и простую фазу слияния. В худшем случае выполненная работа эквивалентна работе при сортировке выбором. Производительность быстрой сортировки существенно зависит от выбора точки разбиения.

6. Алгоритмы поиска

6.1. Последовательный поиск

Задача поиска. Пусть заданы линейные списки: список элементов B=<K1, K2, K3,..., Kn> и список ключей V (в простейшем случае это целые числа). Требуется для каждого значения Vi из V найти множество всех совпадающих с ним элементов из B. Чаще всего встречается ситуация, когда V содержит один элемент, а в B имеется не более одного такого элемента.

Эффективность некоторого алгоритма поиска A оценивается максимальным Max{A} и средним Avg{A} количествами сравнений, необходимых для нахождения элемента V в B. Если Pi – относительная частота использования элемента Ki в B, а Si – количество сравнений, необходимое для его поиска, то

$$Max\{A\} = \max\{ S_i, i=1,n \} \quad ; \quad Avg\{A\} = \sum_{i=1}^{n} P_i S_i .$$

Библиотека БГУИР

Последовательный поиск предусматривает последовательный просмотр всех элементов списка B в порядке их расположения, пока не найдется элемент, равный V. Если достоверно неизвестно, что такой элемент имеется в списке, то необходимо следить за тем, чтобы поиск не вышел за пределы списка, что достигается использованием стоппера.

Очевидно, что Max{A} последовательного поиска равен N. Если частота использования каждого элемента списка одинакова, т.е. P=1/N, то Avg последовательного поиска равно N/2. При различной частоте использования элементов Avg можно улучшить, поместив часто встречаемые элементы в начало списка.

Пример 15.7

Пусть во входном потоке задано 5 целых чисел K1, K2,... K5 и ключ V. Составим программу для последовательного хранения элементов Ki и поиска среди них элемента, равного V, причем такого элемента может и не быть в списке. Без использования стоппера программа может быть реализована следующим образом:

```
#include <stdio.h>
void main(void)
{
   int k[5],v,i,count=0;
   puts("Vvedite elementi:");
   for (i=0;i<5;i++)
   scanf("%d",&k[i]);
   puts("vvedite element dlja poiska");
   scanf("%d",&v);
   i=0;
   while(i<5)
  {        if (k[i]==v)
     {
     printf("element-%d position-%d\n",v,(i+1));
     count++;
     }
              i++;
  }
  if(count==0) printf("element-%d ne naiden ",v);
 }
```


Библиотека БГУИР

6.2. Поиск элемента по индексу

Пример 15.8

```
#include <stdio.h>
void main(void)
{     int k[5],v,i,count=0;
      puts("Vvedite elementi:");
  for (i=0;i<5;i++)
  scanf("%d",&k[i]);
  puts("vvedite element dlja poiska");
      scanf("%d",&v);
      i=0;
      while(i<5)
  {
      if (k[i]==v)
            {
              printf("element-%d position-%d\n",v,(i+1));
              count++;
            }
      i++;
  }
  if(count==0) printf("element-%d ne naiden ",v);
  puts("Vvedite index elementa");
  scanf("%d",&v);
  if(v>=0&&v<5)
  printf("element s indexom %d raven = %d\n",v,k[v]);
  else printf("Elementa s takim indexom net\n");
}
```

Анализ линейного поиска

Линейный поиск – это самый простой тип поиска, потому что при его выполнении просто просматривается множество элементов данных и эти элементы сравниваются с искомыми данными, пока не обнаружится совпадение. Линейный поиск прост в реализации, но не всегда эффективен.

6.3. Бинарный поиск

Для упорядоченных линейных списков существуют более эффективные алгоритмы поиска, хотя и для таких списков применим последовательный поиск. Бинарный поиск состоит в том, что ключ V сравнивается со средним

Библиотека БГУИР

элементом списка. Если эти значения окажутся равными, то искомый элемент найден, в противном случае поиск продолжается в одной из половин списка.

Нахождение элемента бинарным поиском осуществляется очень быстро. Max бинарного поиска равен log2(N), и при одинаковой частоте использования каждого элемента Avg бинарного поиска равен log2(N). Недостаток бинарного поиска заключается в необходимости последовательного хранения списка, что усложняет операции добавления и исключения элементов.

Пример 15.9

```
#include <stdio.h>
void main()
{
    int k[6];
    int v,  i,  j, m;
    for (i=0;i<6;i++)
    scanf("%d",&k[i]);
    scanf("%d",&v);
    i=0;
    j=6;
    m=3;
    while (k[m]!=v)
    {
      if (k[m] < v)
          i+=m;
     else j=m-i;
      m=(i+j)/2;
    }
    printf("%d  %d",v,m);
    }
```

Варианты индивидуальных заданий

Разработать программу для создания пяти динамических массивов: a[ ], b[ ], c[ ], d[ ] и e[ ]. Эти массивы необходимо отсортировать соответственно методом обмена, методом выбора, методом вставки, методом Шелла и методом Хоора. Организовать последовательный поиск определенного элемента в каждом исходном массиве и подсчитать количество совпадений этого элемента в каждом массиве. Организовать бинарный поиск определенного элемента в

Библиотека БГУИР

каждом отсортированном массиве. Разработать в программе меню, структура которого следующая:

Menu:
1. Initilization arrays (инициализация массивов)
2. Result of bubble sort
3. Result of min sort
4. Result of insert sort
5. Result of Shell sort
6. Result of Hoor sort
7. Poisk
   7.1. Posledovatel’nyi poisk
   7.2. Binarnyi poisk

Результаты всех сортировок и поиска записать в файл.

Пример программы

Пример, реализующий алгоритмы сортировки и запись результатов в файл.

```
#include <stdio.h>
#include <iostream.h>
void swap(int *x,int *y);
void bubble_sort(int *a1,int count); //функция пузырьковой сортировки
void min_sort(int *a1,int count); //функция сортировки методом выбора
void insert_sort(int *a1,int count); // функция сортировки методом вставки
void shell_sort(int *x,int count); //функция сортировки методом Шелла
void hoare_sort(int *x,int ,int ); // функция сортировки методом Хоара
void vvod(int *a1,int count); //функция ввода значений массива
void out(int *a1,int count); //функция вывода значений массива
void file_write(int *a1,int count); // функция для записи в файл

FILE *file;
int i,j;
int step=0;

void main(void)
{
int *a,n;
file=fopen("result_sort.txt","w");
puts("Enter length of array:");
scanf("%d",&n);
```

Библиотека БГУИР

```
a=new int[n];
puts("Enter elements of array");
vvod(a,n);
file_write(a,n);
step++;
puts("Sorted array by method \\Bubble Sort\\:");
bubble_sort(a,n);
out(a,n);
file_write(a,n);
step++;
puts("Sorted array by method \\Min element\\:");
min_sort(a,n);
out(a,n);
file_write(a,n);
step++;
puts("Sorted array by method \\Insert Sort\\:");
out(a,n);
file_write(a,n);
step++;
puts("Sorted array by method \\Shell Sort\\:");
shell_sort(a,n);
out(a,n);
file_write(a,n);
step++;
puts("Sorted array by method \\Hoare Sort\\:");
hoare_sort(a,0,n-1);
out(a,n);
file_write(a,n);
fclose(file);
delete []a;
}
void swap(int *x,int *y)
{
int t;
t=*x;
*x=*y;
*y=t;
}
```

Библиотека БГУИР

```
void vvod(int *a1,int count)
{
for(i=0;i<count;i++)
cin>>*(a1+i);
}
void out(int *a1,int count)
{
for(i=0;i<count;i++)
       cout<<*(a1+i)<<" ";
cout<<endl;
}
void bubble_sort(int *a1,int count)
{
       for (i=0; i<count; i++)
       for (j=count-1; j>i; j--)
       if (*(a1+j-1) > *(a1+j))
       swap((a1+j-1),(a1+j)); /*поменять их местами */
}
void min_sort(int *a1,int count)
{
       int k;
       for (i=0; i<count-1; i++)
  {
     for (k=i, j=i+1; j<count; j++) // находим в цикле
       if (*(a1+j)<*(a1+k))
{              // минимальный элемент
         k=j;          // запоминаем его номер в k
         swap((a1+k),(a1+j));
}
  }            // меняем местами минимальный и
          // элем, с которого начинался цикл
}
void insert_sort(int *a1,int count)
{
for (i=1; i<=count-1; i++)
{
       j=i;
       while (*(a1+j)<*(a1+j-1) && j>=1)
    {
```

Библиотека БГУИР

```
            swap((a1+j),(a1+j-1));
            j--;
    }
}}
void shell_sort(int *x,int count)
{   int i,j;     //две переменные цикла
    int gap;     //шаг сортировки
    int sorted;  //флаг окончания этапа сортировки
    for(gap=count/2;gap>0;gap/=2)//начало сортировки
    do  {
          sorted = 0;
          for(i=0,j=gap;j<count;i++,j++)
            if(*(x+i)>*(x+j))
             {   swap((x+i),(x+j));
                 sorted = 1;
             }
        }while(sorted);
}
void hoare_sort(int *x,int l,int r)
{
int sr=*(x+(l+r)/2);
i=l;
j=r;
do
{  while(*(x+i)<sr) i++;
   while(*(x+j)>sr) j--;
   if(i<=j)
   {  swap((x+i),(x+j));
      i++;    j--;
   }
}while(i<=j);
if(i<r)
   hoare_sort(x,i,r);
if(j>l)
   hoare_sort(x,l,j);
}
void file_write(int *a1,int count)
{  if(step==0)
   fprintf(file,"\t\tRESULTS OF SORTS\n");
```


Библиотека БГУИР

```
if(step==1)
   fprintf(file,"1.Bubble Sort\n");
if(step==2)
   fprintf(file,"\n2.Min Sort\n");
if(step==3)
   fprintf(file,"\n3.Insert Sort\n");
if(step==4)
   fprintf(file,"\n4.Shell Sort\n");
if(step==5)
   fprintf(file,"\n5.Hoare Sort\n");
if(step!=0)
for(i=0;i<count;i++)
   fprintf(file,"%d ",*(a1+i));
}
```

ЛАБОРАТОРНАЯ РАБОТА №16

ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

Цель: Освоить практическое использование численных методов, научиться вычислять интеграл функции, осуществлять сравнение методов расчета интеграла и их погрешности.

Теоретические сведения

При решении задач, связанных с вычислением интеграла, отсутствует возможность осуществлять преобразование функции в соответствии со специальными функциями и правилами, принятыми в математике. На практике эта задача сводится к нахождению площади фигуры, образованной (ограниченной) данной функцией, осью координат, прямыми x = a и x = b, где a и b – крайние точки (рис. 16.1).

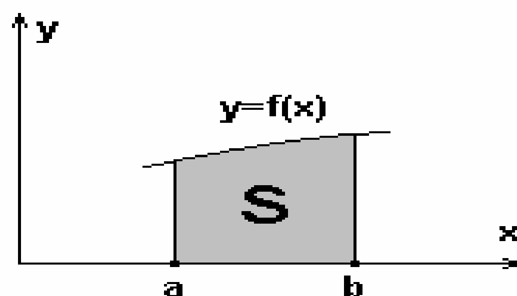


Рис. 16.1. Интеграл функции f(x) от a до b

Библиотека БГУИР

При расчете площади данной фигуры применяют метод разбивки данной фигуры на множество элементов (прямоугольников или трапеций) с очень маленькой шириной, при этом можно предположить, что функция (верхняя часть этих элементов) представляет собой прямую (рис. 16.2). Подобную методику называют аппроксимацией.

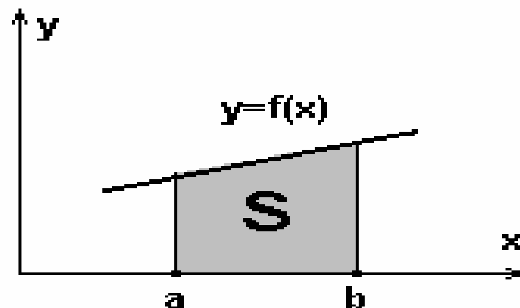


Рис. 16.2. Аппроксимация функции f(x)

Выделяют два наиболее простых способа расчета интеграла: метод прямоугольника (метод Симпсона) и метод трапеции. В свою очередь метод Симпсона делится на три варианта: левого прямоугольника, правого и серединного.

1. Методы Симпсона

1.1. Метод левого прямоугольника заключается в том, что площадь прямоугольника вычисляется на основании левой стороны прямоугольника и ширины прямоугольника – шага аппроксимации (рис.16.3).

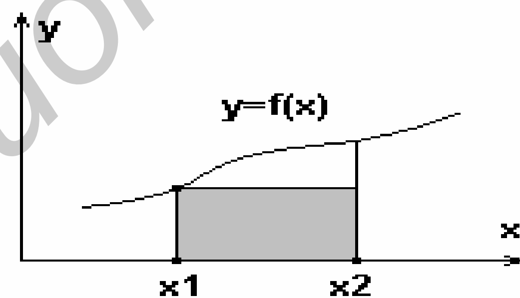


Рис. 16.3. Метод левого прямоугольника

Математически это можно записать следующим образом

$$dS = f(x1)*dx,$$

где dS – приращение площади – площадь маленького прямоугольника;

dx – приращение координат (x);

f(x1) – левая сторона прямоугольника.

Библиотека БГУИР

1.2. Метод серединного прямоугольника заключается в том, что площадь прямоугольника вычисляется на основании значения функции в середине прямоугольника и ширины прямоугольника – шага аппроксимации (рис. 16.4).

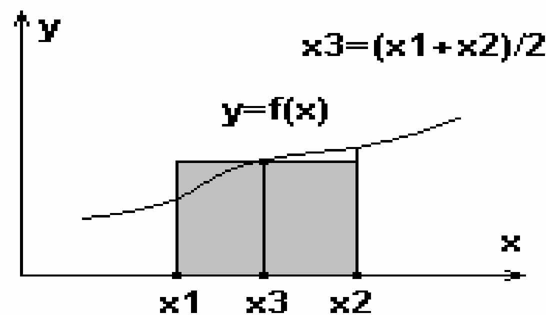


Рис. 16.4. Метод серединного прямоугольника

Математически это можно записать как

$$dS = f((x1+x2)/2)*dx,$$

где dS – приращение площади – площадь маленького прямоугольника;

dx – приращения координат (x);

f(x1) – левая сторона прямоугольника;

f(x2) – правая сторона прямоугольника.

1.3. Метод правого прямоугольника заключается в том, что площадь прямоугольника вычисляется на основании правой стороны прямоугольника и ширины прямоугольника – шага аппроксимации (рис. 16.5).

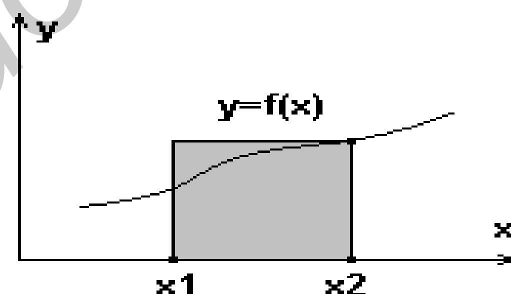


Рис. 16.5. Метод правого прямоугольника

Математически это можно записать как

$$dS = f(x2)*dx,$$

где dS – приращение площади;

dx – приращения координат (x);

f(x2) – правая сторона прямоугольника.

Библиотека БГУИР

2. Метод трапеции

Данный метод заключается в том, что площадь трапеции вычисляется на основании произведения среднеарифметического значения левой и правой сторон прямоугольника и ширины прямоугольника – шага аппроксимации (рис. 16.6).

Математически это можно записать как

$$dS = (f(x1)+f(x2))/2 * dx,$$

где dS – приращение площади;

dx – приращение координат (x);

f(x1) – левая сторона прямоугольника;

f(x2) – правая сторона прямоугольника.

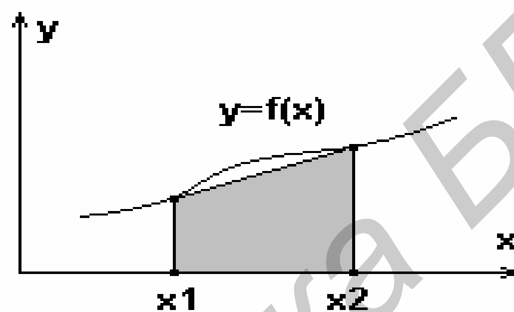


Рис. 16.6. Метод трапеции

Варианты индивидуальных заданий

Общее задание. Рассчитать интеграл функции всеми четырьмя способами в соответствии с вариантом индивидуального задания. Найти среднеарифметическое значение интеграла на основе полученных значений. Найти максимальную погрешность методов относительно среднеарифметического значения. Шаг аппроксимации лучше всего выбирать равным 0,0001.

Требования к программе. Реализовать в программе меню, позволяющее осуществить следующие действия: ввод исходных данных, расчет интеграла методом левого, правого, серединного треугольника, методом трапеции; вывод всех результатов в виде таблицы (исходные и расчетные данные); запись этой таблицы в файл; чтение ее из файла; выход из программы. Все логически законченные действия оформить в виде отдельных функций.

Варианты заданий:

1. f(x) = 3sin(x2) на интервале от 0,1 до 0,3.
2. f(x) = cos(2x3) на интервале от 0,2 до 0,4.
3. f(x) = 4tg (x2) на интервале от 0,1 до 0,2.
4. f(x) = 2ctg(x2) на интервале от 0,1 до 0,3.

Библиотека БГУИР

5. $f(x) = ctg(2x)/2$ на интервале от 0,1 до 0,2.
6. $f(x) = tg(3x)$ на интервале от 0,05 до 0,1.
7. $f(x) = ctg(\sin x)$ на интервале от 0,15 до 0,2.
8. $f(x) = 2ctg(x/2)$ на интервале от 0,15 до 0,2.
9. $f(x) = ctg(x3)/3$ на интервале от 0,1 до 0,3.
10. $f(x) = x \sin(x2/2)$ на интервале от 0,3 до 0,4.
11. $f(x) = x \cos(x/3)$ на интервале от 0,2 до 0,3.
12. $f(x) = x \cos(x2)$ на интервале от 0,1 до 0,3.
13. $f(x) = \cos(x5)$ на интервале от 0,1 до 0,3.
14. $f(x) = x \sin(x2)$ на интервале от 0,1 до 0,3.
15. $f(x) = x\ sinxcos(x2)$ на интервале от 0,1 до 0,3.

Библиотека БГУИР

Литература

1. Березин, Б. И. Начальный курс С и С++ / Б. И. Березин, С. Б. Березин. – М. : Диалог-МРТИ, 1999.

2. Керниган, Б. Язык программирования С / Б. Керниган, Д. Ритчи. – М. : Финансы и статистика, 1992.

3. Касаткин, А. И. Профессиональное программирование на языке Си : от Turbo C к Borland C++ : справ. пособие / А. И. Касаткин, А. Н. Вольвачев. – Минск : Выш. шк., 1992.

4. Страуструп, Б. Язык программирования С++. – 2-е изд. В 2 т. / Б. Страуструп. – Киев : Диа Софт, 1993.

5. Фьюэр, А. Задачи по языку Си / А. Фьюэр. – М. : Финансы и статистика, 1985.

6. Хэнкок, Л. Введение в программирование на языке Си / Л. Хэнкок, М. Кригер. – М. : Радио и связь, 1986.

7. Бери, В. Язык Си : введение для программистов / В. Берри, Б. Микинз. – М. : Финансы и статистика, 1988.

8. Уэйт, М. Язык Си. Руководство для начинающих / М. Уэйт, С. Прама, Д. Мартин. – М. : Мир, 1988.

9. Больски, М. Н. Язык программирования Си : справочник / М. Н. Больски. – М. : Радио и связь, 1988.

10. Юлин, В. А. Приглашение к Си / В. А. Юлин, И. Р. Булатова. – Минск : Выш. шк., 1990.

11. Уингер, Р. Язык Турбо Си / Р. Уингер. – М. : Мир, 1991.

12. Романовская, Л. М. Программирование в среде Си для ПЭВМ ЕС / Л. М. Романовская, Т. В. Русс, С. Г. Свитковский. – М. : Финансы и статистика, 1992.

Библиотека БГУИР

Св. план 2007, поз. 141

Учебное издание

Живицкая Елена Николаевна
Комличенко Виталий Николаевич
Кардаш Сергей Николаевич и др.

ОСНОВЫ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ

Лабораторный практикум

для студентов специальности I-40 01 02-02
«Информационные системы и технологии в экономике»
всех форм обучения

В 2-х частях

Часть 2

Библиотека БГУИР

Редактор Т. П. Андрейченко
Корректор М. В. Тезина

Подписано в печать 04.03.2007. Формат 60х84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 6,51.
Уч.-изд. л. 6,1. Тираж 250 экз. Заказ 33.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки, 6