МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Кафедра экономической информатики

А.В.Бахирев, Е.Н. Живицкая,
В.Н. Комличенко, С.А. Соколов

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
по курсу "Основы информатики и вычислительной техники"

для студентов экономической специальности
В 2 частях.
Часть 1.

МИНСК 2000

УДК 002.5 +681.3 (075.8)
ББК 22.1 Я 73
Л 12

Лабораторный практикум по курсу "Основы информатики и вычислительной
Л 12 техники" для студентов экономической специальности. В 2 ч. Ч. 1.
А.В.Бахирев, Е.Н. Живицкая, В.Н. Комличенко, С.А. Соколов. -Мн.: БГУИР, 2000.- с.48

ISBN 985-444-184-9 (ч.1)

В практикуме представлен курс из 8 лабораторных работ, даны краткие теоретические сведения, примеры и варианты задания для лабораторных работ.

УДК 002.5 +681.3 (075.8)
ББК 22.1 Я 73

ISBN 985-444-185-7
985-444-184-9 (ч.1)

Лабораторная работа №1
Операторы ввода-вывода и языковые средства ветвления

1.Элементы программирования, общие понятия
1.1.Понятие идентификатора

Идентификатор используется в качестве имени объекта (функции, переменной, константы и т.п.). Идентификаторы должны выбираться с учетом следующих правил:
1) обязательно начинаться с буквы латинского алфавита (a,..., z. A,..., Z) или с символа подчеркивания (_), в них могут использоваться буквы латинского алфавита, символ подчеркивания и цифры (0,...,9) . Использование других символов в идентификаторах запрещено;
2) буквы нижнего регистра (a, ..., z), применяемые в идентификаторах, отличаются от букв верхнего регистра (A, ..., Z). Это означает, что следующие идентификаторы считаются разными: prog, ProG, PROG, pRoG и т.п.;
3)идентификаторы могут включать любое число символов, из которых воспринимаются и используются для выявления различных объектов (имен) только первые 32.

1.2.Типы данных и объявление переменных

Программа оперирует с различными данными, которые могут быть простыми и структурированными. Простые данные - это целые и вещественные числа, текст и указатели (содержат адреса памяти, по которым размещаются данные). В языке различают понятия описание переменной и ее определение (объявление). Описание устанавливает свойства объекта: его тип, размер и т.д. Определение наряду с этим вызывает выделение памяти. Каждый тип данных определяется ключевыми словами, которые приведены в табл.1.1:

Таблица 1.1

Название типа	Тип значения переменной	Диапазон значений	Необходимая память, в битах	Примечания
1	2	3	4	5
Int	Целый	-32768 ...32767	16	Задает значения, к которым относятся все целые числа, например –6, 0, 28 и т.д.

Окончание табл.1.1 .

1	2	3	4	5
short	Короткий и целый	-32768 ...32767	16	Объекты short не могут быть больше, чем int. В Borland C int и short равной длины
long	Длинный и целый	214748364… 2147483647	32	Используется, когда диапазон значений выходит за пределы диапазона типа int
char	Сим-вольный	Символы кодовой таблицы ASCII (0...255)	8	Задает значения, которые представляют различные символы, например, w, у, ф, 4, !, ., * и т. д. Этот тип часто используется как наименьшее беззнаковое целое значение
unsigned	Беззнако-вый			Модификатор типов char, short, int, long, определяющий их беззнаковыми.[1)]
float	Вещест-венный	±3.4e-38… ±3.4e+38	32	Определяет вещественные числа, дробная часть которых отделяется точкой (например, -5.27, 0.0, 31.69 и т.д.). Вещественные числа могут записываться в экспоненциальной форме. Например: -1.58e+2 (что равно -1,58 * 10^2), 3.61e-4 (что равно 3,61 *10^{-4}).
double	Вещест-венный, двойная точность	±1.7e-308... ±1.7e+308	64	Определяет вещественные переменные двойной точности, занимающие в два раза больше памяти, чем переменная типа float

[1)]Типы unsigned (например unsigned char) могут принимать большие по абсолютной величине положительные значения, чем переменные знаковых типов, за счет использования знакового бита для представления числа. Например, переменная типа unsigned int может принимать значения от 0 до 65535 (просто int при том же размере в битах от –32768 до 32767). По умолчанию unsigned a определяется как unsigned int a.

Примеры объявления данных:

int a, b;
unsigned i, j;
float k;

Здесь объявлены переменные: целые a и b , беззнаковые целые i и j , вещественное число одинарной точности k.

2.Ввод – вывод информации

В языке Си имеется ряд функций, предназначенных для реализаций операций ввода-вывода. Наиболее используемая – функция форматированного вывода:
printf(“управляющая строка вывода“, список_переменных_через_запятую);

Формат printf включает в себя как текстовые сообщения, так и управляющие символы. Управляющим символам предшествует символ %, за которым могут следовать буквы, определяющие прототип вывода значений переменных. Выбор прототипа зависит от типа переменной, значение которой будет выводиться вместо прототипа. Основные прототипы переменных перечислены в табл.2.1.

Таблица 2.1

Название типа	Формат	Примечание
char	%c	
char[n]	%s	(Строка - массив символов), где n – количество символов в строке.
int	%d	
long	%ld	
float	%f	
double	%lf	

Количество форматов в маске ввода должно соответствовать количеству переменных в списке переменных после кавычек. Переменные разделяются между собой запятыми. В формат могут входить также специальные символы, приведенные в табл.2.2.

Таблица 2.2.

Символ	Назначение
\n	Новая строка
\t	Табуляция
\\	Вывод символа \
\”	Вывод символа “

Символы, не являющиеся символами формата или спецсимволами, непосредственно выводятся функцией printf.

Пример использования оператора printf для вывода значений переменных a,b:

```
int a,b;                    // объявление переменных a,b
```

```
printf("a = %d , b = %d;\n", a, b);          // вывод значений переменных a,b
                                              //в форме a=5, b=10;
```

Оператор ввода предназначен для ввода значений переменных с клавиатуры. Формат оператора scanf соответствует формату оператора printf. Отличие заключается в том, что перед значениями переменных всех типов, за исключением массивов (строк символов), ставится амперсант – символ “&.”. Он означает, что в распоряжение функции предоставляется не содержимое, а адрес переменной, что будет рассмотрено в разделе изучения указателей.

scanf(“формат“,X1,…Xn);

Пример использования оператора scanf для ввода значений переменных a,b целого типа:

```
int a,b;                                  // объявление переменных a,b
scanf ( "%d%d", &a, &b );                 // ввод значений переменных a,b клавиатуры
printf("a = %d  b = %d\n", a, b);         // вывод значений переменных a,b
```

3.Языковые средства ветвления

Все выражения, реализующие условия в конструкции выбора, должны заключаться в круглые скобки.

Логические операции. В языке Си для работы с логическими операторами приняты несколько основных вариантов обозначения операций сравнения, которые представлены в табл.3.1.

Таблица 3.1

Обозначение	Операция
!=	Не равно
==	Равно
<	Меньше
>	Больше
<=	Меньше равно
>=	Больше равно
&&	Логическое И (исполняется, если все условия выполнены)
||	Логическое ИЛИ (исполняется, если хотя бы одно условие выполнено)

Если необходимо проверять несколько условий, то каждое условие берется в свои скобки, а между скобками ставятся логические операторы (в зависимости от логики). В случае выполнения нескольких условных операторов эти операторы берутся

в фигурные скобки. Обратите внимание, что перед оператором не ставится точка с запятой.

Секция выполняется каждый раз при проходе цикла.

3.1.Оператор if

Синтаксис оператора if:

if (выражение) оператор;

Там, где синтаксис языка предписывает использовать оператор, может стоять и составной (блок операторов, заключенный в фигурные скобки), и пустой оператор (символ «;» - точка с запятой). Если выражение в заголовке условного оператора вырабатывает ненулевое значение, то оператор в условном операторе выполняется, в противном случае управление передается оператору, следующему за условным. Пример:

```
int a;                          // объявление переменных a
scanf ( "%d", &a);              // ввод значений переменных a с клавиатуры
if(a==3)                        // сравнение переменной a с 3
        printf( "a равно 3");   // вывод сообщения на экран в случае
                                // выполнения условия
```

3.2.Конструкция if else

Синтаксис оператора if else таков:

```
if ( выражение )
оператор1;
else
оператор2;
```

Если значение выражения не равно нулю, то выполняется оператор1, в противном случае - оператор2. Пример:

```
int a;                          // объявление переменных a
scanf ( "%d", &a);              // ввод значений переменных a с клавиатуры
if(a==3)                        // сравнение переменной a с 3
printf( "a равно 3");           // вывод сообщения на экран в случае
                                    // выполнения условия
else printf( "a не равно 3");   // вывод сообщения на экран в случае
                                  // невыполнения условия
```

3.3.Условная операция ?

Условная операция **?** может с успехом использоваться вместо конструкции if else там, где входящие в нее операторы являются простыми выражениями. Синтаксис условной операции таков:

результат = выражение ? выражение1 : выражение2;

Для примера рассмотрим программу. Переменной result при ее инициализации будет присвоено значение b, если выражение (a < 0) истинно, и a, если выражение (a < 0) ложно. В примере значение переменной result зависит от введенного значения переменной a:

```
int a,b=0;                                  // объявление переменных a
scanf ( "%d", &a);                          // ввод значений переменных a с
                                            // клавиатуры
int result = ( a < 0 ) ? b : a;             // объявление переменной result
                                            // по условию
printf( "a = %d  b = %d result = %d\n", a, b, result ); // вывод значений
                                            // переменных a,b, result
```

3.4.Оператор switch

Конструкция switch заменяет разветвленный многократный оператор if else. Синтаксис оператора switch таков:

```
switch ( выражение ) {
        case константное_выражение_1 :
                оператор(ы) ; break;
        case константное_выражение_2 :
                оператор(ы) ; break;
        case константное_выражение_3 :
                оператор(ы) ; break;
        default:
                оператор(ы) ; break;
}
```

После вычисления выражения в заголовке оператора его результат последовательно сравнивается с константными выражениями, начиная с самого верхнего, пока не будет установлено их соответствие. Тогда выполняются операторы внутри соответствующего case, управление переходит на следующее константное выражение, и проверки продолжаются. Именно поэтому в конце каждой последовательности операторов должен присутствовать оператор break. После выполнения последовательности операторов внутри одной ветки case, завершающейся

оператором break, происходит выход из оператора switch. Обычно оператор switch используется тогда, когда программист хочет, чтобы была выполнена только одна последовательность операторов из нескольких возможных.

Каждая последовательность операторов может содержать нуль или более отдельных операторов. Фигурные скобки в этом случае не требуются.

Ветка, называемая default (умолчание), может отсутствовать. Если она есть, то последовательность операторов, стоящая непосредственно за словом default и двоеточием, выполняется только тогда, когда сравнение «выражение» ни с одним из стоящих выше константных выражений (в case) не истинно. Пример:

```
int a;                                  // объявление переменных a
scanf ( "%d", &a);                      // ввод значений переменных
                                            // a и с клавиатуры
switch(a)
{       case 3: printf( "a равно 3");   // вывод сообщения на экран
                break;                  // в случае a=3
        case 4: printf( "a равно 4");   // вывод сообщения на экран
                break;                  // в случае a=4
        default:
                printf( "a = %d\n", a); // вывод значения переменной a
}
```

Варианты задания

1.Разработать программу определения типа самолета для перевозки каких-либо грузов. В программе вводится вес груза и на его основании осуществляется выбор типа самолета. Груз менее 5т способен перевозит "ТУ-134". Вес более 5т, но менее 30т- "ИЛ-86". Вес более 30т, но менее 45т - "Руслан".

2. Разработать программу определения величины заработной платы от категории сотрудника. В программе вводится номер категории и на ее основании осуществляется выбор величины заработной платы. Для 1-й категории зарплата составляет 100 дол., 2-й - 80, 3-й - 60.

3. Разработать программу определения платы за пересылку писем. В программе вводится расстояние до адресата. Если оно менее 100 км, цена составляет 100 тыс. р., свыше 100, но менее 1000 - 200 тыс.р., свыше 1000 -300 тыс.р.

4. Разработать программу определения количества дополнительных дней к отпуску. В программе вводится количество лет, отработанных на предприятии. Если отработано менее 5 лет, то дополнительных дней к отпуску не предусмотрено. Если от 5 до 15 - плюс 2 дня к отпуску. Если свыше 15 - плюс 5 дней к отпуску.

5.Разработать программу определения скидки на оплату туристического тура в зависимости от срока предварительной оплаты. В программе вводится, за какое

количество дней производится оплата. При оплате за 90 дней скидка 15%, при 60 - 10%, при 30 -5%.

6.Разработать программу определения оплаты международных разговоров в зависимости от времени суток. В программе вводится время звонка (час - от 0 до 24). При определении оплаты исходить, что звонок с 0:00 до 7:00 стоит 1дол/мин, с 7.00 до 18:00 - 3 дол/мин, 18:00 до 24:00 - 2 дол/мин.

7.Разработать программу определения стоимости проезда в электричке в зависимости от зон. В программе вводится количество зон, пересекаемых электричкой при перевозке. Для одной зоны - 100 р., для 2-х - 200 р., для 3-х - 300 р.

8.Разработать программу определения степени административного наказания от числа повторных нарушений. В программе вводится номер повторного нарушения. При втором нарушении выводится сообщение - "штраф в размере 5 минимальных зарплат", при третьем - "год условно", при четвертом - "год исправительных работ".

9.Разработать программу начисления премиальных в зависимости от процента перевыполнения плана. В программе вводится процент перевыполнения плана. При перевыполнении плана на 10% - премия 1000 р., 20% - 2000 р., свыше 30% - 4000 р.

10.Разработать программу определения величины подоходного налога в зависимости от величины зарплаты. В программе вводится сумма зарплаты. При сумме менее на 20 тыс.р. ставка налога 10%, при сумме более 20 тыс.р. но менее 60 тыс.р. - 15%, свыше 120 тыс.р. -20%.

11.Разработать программу определения скидки при оплате за коммунальные услуги. В программе вводится возраст лица, за которое осуществляется оплата. Для детей (возраст от 0 до 16 лет) - 70%, для пенсионеров (от 55 лет) -60%, 100% для остальных категорий граждан.

12.Разработать программу контроля веса фасовочного аппарата, который фасует пищевые изделия по одному килограмму. В программе дан точный вес контрольных весов. При превышении веса более чем на 10г - выдается сообщение "аппарат требует внеочередной наладки", при недовесе более 5г - выдается сообщение "вернуть для перефасовки", в противном случае - "годен для реализации".

13.Разработать программу определения степени разрушений при землетрясении. В программе вводится сила землетрясения в баллах от 1 до 10. При силе землетрясения менее 3 баллов выводится сообщение "слабые разрушения", от 4 до 6 - "сильные разрушения", от 7 до 10 - "полное разрушение".

14.Разработать программу определения размера штрафа при нарушении правил дорожного движения. В программе вводится серьезность нарушения (1- незначительное нарушение, 2 - серьезное нарушение, 3 - грубое нарушение). При незначительном нарушении штраф - 1000 р., при серьезном нарушении - 3000 р., при грубом нарушении - 8000 р.

15.Разработать программу определения стоимости проживания в гостиничном номере. В программе вводится количество "звездочек" в разряде гостиницы и на его основании осуществляется выбор стоимости оплаты. В 3-звездочной гостинице номер стоит 500 р/день, в 4-зведочной- 1000 р/день, в 5-звездочной – 2000 р/день.

16.Разработать программу определения стоимости компьютера от номера версии процессора «Pentium». В программе вводится номер процессора. Для компьютеров с процессором «Pentium» – 200 дол., для «Pentium II» – 300 дол. И для «Pentium III» – 400 дол.

Внимание. Во всех вариантах осуществить вывод результата согласно формату, указанному преподавателем.

Лабораторная работа №2
Циклы и массивы

1.Циклы

Циклы, или итерационные структуры, позволяют повторять выполнение отдельных операторов или групп операторов. Число повторений в некоторых случаях фиксировано, а в других определяется в процессе счета на основе одной или нескольких проверок условий.

Циклы завершаются в следующих случаях:

1)обратилось в нуль условное выражение в заголовке цикла.;

2)в теле цикла выполнился оператор break;

3)в теле цикла выполнился оператор return.

В первых двух случаях управление передается на оператор, располагающийся непосредственно за циклом. В третьем случае происходит возврат из функции (завершение работы данной функции).

Бывают циклы с проверкой условия перед началом выполнения тела цикла (top-testing), по окончании выполнения тела (bottom-testing) или внутри тела (middle-testing). Ниже рассмотрены все указанные типы циклов.

1.1.Цикл while

Синтаксис цикла while (пока):

while (условное_выражение) оператор

Ясно, что в цикле типа while проверка условия производится перед выполнением тела цикла (оператор). Если результат вычисления условного выражения не равен нулю, то выполняется оператор (или группа операторов). Перед входом в цикл while в первый раз обычно инициализируют одну или несколько переменных для того, чтобы

условное выражение имело какое-либо значение. Оператор или группа операторов, составляющих тело цикла, должны, как правило, изменять значения одной или нескольких переменных, входящих в условное выражение, чтобы в конце концов выражение обратилось в нуль, и цикл завершился.

Потенциальной ошибкой при программировании цикла while, как, впрочем, и цикла любого другого типа, является запись такого условного выражения, которое никогда не прекратит выполнение цикла. Такой цикл называется бесконечным (например цикл: while (a) printf("Circle"), где a - любое число, отличное от 0. Цикл будет бесконечно выводить на экран дисплея текст Circle) . Пример:

```
#include <stdio.h>                  // подключение библиотеки stdio.h
void main(void)                     // основная функция main
{       int a;                      // объявление переменных a
        scanf ( "%d", &a);          // ввод значений переменных a и с клавиатуры
        while(a>=0)                 // цикл повторяется пока a>=0
        { printf( "a = %d\n", a);   // вывод значения переменной a
            a--;                    // уменьшение значения переменной a на один
        }
}
```

1.2.Цикл do while

В цикле do while проверка условия осуществляется после выполнения тела цикла. Синтаксис цикла:

```
do  оператор;
        //тело цикла
while ( условное_выражение )
```

В языке Си вместо одиночного оператора (например в теле рассматриваемого цикла) может быть подставлена группа операторов (блок). Цикл while прекращает выполняться, когда условное выражение обращается в нуль (становится ложным). Пример:

```
#include <stdio.h>                  // подключение библиотеки stdio.h
void main(void)                     // основная функция main
{       int a;                      // объявление переменных a
        scanf ( "%d", &a);          // ввод значений переменных a и с клавиатуры
        do{                         // начало цикла
            printf( "a = %d\n", a); // вывод значения переменной a
            a--;                    // уменьшение значения переменной a на 1
        } while(a>=0);              // цикл повторяется пока a>=0
}
```

1.3.Цикл for

Наиболее общей формой цикла в языке С является цикл for. Цикл for - это более общая и более мощная форма, чем аналогичный цикл в языках Паскаль и Бейсик. Конструкция for выглядит следующим образом:

for (выражение1; выражение2; выражение 3) оператор;

Каждое из трех выражений можно опускать. Хотя в принципе каждое из этих выражений может быть использовано программистом как угодно, обычно первое выражение служит для инициализации индекса, второе -для выполнения проверки на окончание цикла, а третье выражение - для изменения значения индекса. Формально это правило можно описать так:

1. Если первое выражение присутствует, то оно вычисляется.
2. Вычисляется второе выражение (если оно присутствует). Если вырабатывается значение 0, то цикл прекращается, в противном случае цикл будет продолжен.
3. Исполняется тело цикла.
4. Вычисляется третье выражение (если оно присутствует).
5. Выполняется переход к п.2.

Выполнение в любом месте тела цикла оператора continue приводит к немедленному переходу к шагу 4. Пример:

```
#include <stdio.h>                  //подключение библиотеки stdio.h
void main(void)                     //основная функция main
{       int a;                      //объявление переменных a
        for(a=0; a<10; a++)         //цикл от 0 до 9-ти
        printf( "a = %d\n", a);     // вывод значения переменной a
}
```

Цикл for можно свести к циклу while следующим образом:

Цикл for:

for (выражение1; выражение2; выражение3) оператор;

переводится в:

```
выражение1;
while ( выражение2 ) {
оператор;
выражение3;          }
```

2.Массивы

Массив - это структурированный тип данных, состоящий из нескольких элементов одного и того же типа. В качестве типа массива может быть выбран

любой известный (объявленный) тип языка СИ. Отличие объявления массива от объявления обычной переменной заключается в наличии квадратных скобок после названия массива. В них указывается количества элементов в массиве. Пример объявления массива:

```
int a [100];
char b [30];
float c [42];
```

В приведенном примере массив "a" состоит из 100 целых чисел, массив "b" состоит из 30 символов, массив "c" состоит их 42 вещественных чисел. Нумерация элементов массива начинается с нулевого элемента.

Массивы бывают одномерные и многомерные. Количество измерений массива определяется при декларации по количеству квадратных скобок после имени массива. Пример:

```
int a[10] ;
int b[100][30];
int c[20][78][45];
```

В приведенном примере массив "a" является одномерным, массив "b" - двумерным, массив "c" - трехмерным. Одномерный массив иногда называют вектором. Наиболее часто используются одномерные и двумерные массивы.

Для одномерного массива все элементы хранятся в виде строки. Обращение к массиву осуществляется по его имени. Для обращения к конкретному элементу массива необходимо помимо имени массива задать порядковый номер элемента в массиве (индекс). Например, обращение к девятому элементу запишется так: a[5].

Двумерный массив представляется как одномерный массив, элементы которого тоже массивы. Элементы двумерного массива хранятся по строкам. Если проходить последовательно по элементам массива в порядке их расположения в памяти, то быстрее всего изменяется самый правый индекс. Например, обращение к девятому элементу пятой строки запишется так: a[5][9].

Пусть задано объявление: int a[2][3]; Тогда элементы массива a будут размещаться в памяти следующим образом: a[0][0], a[0][1] a[0][2], a[1][0], a[1][1], a[1][2]. Имя массива a – это указатель константы, которая содержит адрес его первого элемента (для нашего примера – a[0][0]). Предположим, что a=1000. Тогда адрес элемента a[0][1] будет равен 1002 (элемент типа int занимает в памяти 2 байта), адрес следующего элемента a[0] [2] – 1004 и т.п. Что же произойдет, если вы выберете элемент, для которого не выделена память. К сожалению, компилятор не следит за этим. В результате возникнет ошибка, и программа будет работать не верно.В ряде случаев происходит аварийное завершение программы.

Язык С позволяет инициализировать массив при объявлении. Для этого используется такая форма:

тип имя_массива [количество_элементов] = { список значений };

Рассмотрим примеры:

```
int a[5] = {0,1,2,3,4};
char c[7] = { 'a','b','c','d','e','f','g'};
int b [2] [3]= {1,2,3,4,5,6};
```

В последнем случае: b[0][0] = 1, b[0][1] = 2, b[0][2] = 3, b[1][0] = 4, b[1][1] = 5, b[1][2] = 6.

3.Примеры использования циклов для операций с массивами

Обычно последовательный доступ к элементам массива осуществляется при помощи операторов цикла for, while или do while. Проиллюстрируем использование оператора цикла и функции scanf для ввода значений массива int a[3] :

```
for(i=0;i<3;i++)
    scanf("%d",&a[i]);
```

или

```
i=0;
while(i<3){
   scanf("%d",&a[i]);
   i=i+1;
}
```

или

```
i=0;
do{
   scanf("%d",&a[i]);
   i++;
} while(i<=3);
```

Ввод элементов массива можно осуществлять также с помощью оператора cin. Пример:

```
for(i=0;i<3;i++)
   cin>>a[i];
```

Пример вывода элементов массива:

```
for(i=0;i<3;i=i+1)
    printf("%d",a[i]);
```

или

```
for(i=0;i<3;i=i+1)
     cout<<a[i];
```

Для изменения направления ввода\вывода необходимо изменить начальное значение переменной управляющей циклом и знак операции увеличения этой переменной с положительного на отрицательный. Пример:

```
i=3;
while(i>=0){
    i=i-1;
    scanf("%d",&a[i]);
}
```

или

```
for(i=0;i<3;i--)
    cout<<a[i];
```

Для возможности изменения количества обрабатываемых элементов необходимо объявить массив максимально возможной длины. Перед использованием этого массива необходимо будет определить размер используемого фрагмента массива (определить явно присваиванием или ввести с клавиатуры). Размер используемого фрагмента должен быть меньше либо равен размеру объявленного массива.

Внимание! Для использования функций scanf и printf необходимо подключить библиотеку "stdio.h". Для использования операторов cin и cout необходимо подключить библиотеку "iostream.h".

Типичные ошибки в программах:

1) После круглых скобок операторов for и while точка с запятой не ставится.

2) Количество открытых "{" и закрытых "}" скобок в программе и в любой функции должно быть одинаковое. При создании программ существует правило - при открытии какой- либо скобки, она тут же закрывается, а все необходимое содержимое пишется между скобок.

Варианты задания №1

1.Создать массив целых чисел из 10 элементов. Ввести значения элементов массивов с клавиатуры при помощи функции scanf и вывести их на экран в обратном порядке при помощи оператора printf.

2.Создать массив целых чисел из 7 элементов. Задать начальные значения элементов массива при объявлении массива и вывести их на экран при помощи оператора cout и оператора while.

3.Создать массив целых чисел из 8 элементов. Ввести значения элементов массивов с клавиатуры при помощи оператора cin и вывести элемент массива при помощи оператора цикла while и оператора вывода cout.

4.Создать массив целых чисел из 10 элементов. Ввести значения элементов массивов с клавиатуры при помощи оператора cin и вывести элемент массива с

указанием их порядковых номеров в массиве. Для вывода использовать оператор цикла while и функции cout.

5.Создать массив символов из 8 элементов. Задать начальные значения элементов массива прямо при объявлении массива и вывести каждый второй элемент массива на экран при помощи оператора цикла for с использованием функции printf.

6.Создать массив символов из 8 элементов. Ввести значения элементов массивов с клавиатуры при помощи функции cin и вывести каждый второй элемент массива на экран при помощи оператора цикла while с использованием функции cout.

7.Создать массив символов из 12 элементов. Ввести значения элементов массивов с клавиатуры при помощи функции scanf и вывести каждый третий элемент массива на экран при помощи оператора цикла for с использованием функции printf.

8.Создать массив целых чисел из 12 элементов. Задать начальные значения элементов массива прямо при объявлении массива и вывести каждый третий элемент массива на экран при помощи оператора цикла while с использованием функции cout.

9.Создать массив целых чисел из 8 элементов. Ввести значения элементов массивов с клавиатуры при помощи функции scanf и вывести элементы массива на экран, значения которых больше 2 (при вводе элементов массива необходимо вводить значения массива так, чтобы часть значений элементов массива оказывалась больше 2, а часть меньше). Для вывода использовать оператор цикла for и функцию printf.

10.Создать массив вещественных (дробных) чисел из 8 элементов. Ввести значения элементов массивов с клавиатуры при помощи функции cin и вывести элементы массива на экран, значения которых меньше 3 (при вводе элементов массива необходимо вводить значения массива так, чтобы часть значений элементов массива оказывалась больше 3, а часть меньше). Для вывода использовать оператор цикла while и функцию cout.

11.Создать массив целых чисел из 6 элементов. Задать начальные значения элементов массива прямо при объявлении массива и вывести элементы массива на экран в обратном порядке, значения которых больше 4 (при вводе элементов массива необходимо вводить значения массива так, чтобы часть значений элементов массива оказывалась больше 4, а часть меньше). Для вывода использовать оператор цикла for и функцию printf.

12.Создать массив вещественных (дробных) чисел из 6 элементов. Ввести значения элементов массивов с клавиатуры при помощи функции cin и вывести элементы массива на экран в обратном порядке, значения которых меньше 5 (при вводе элементов массива необходимо вводить значения массива так, чтобы часть значений элементов массива оказывалась больше 5, а часть меньше). Для вывода использовать оператор цикла while и функцию cout.

13.Создать массив целых чисел из 7 элементов. Ввести значения элементов массивов с клавиатуры при помощи функции scanf и вывести элементы массива в виде колонки чисел с указанием их порядкового номера элемента массива. Для вывода использовать оператор цикла for и функцию printf.
14.Создать массив вещественных (дробных) чисел из 7 элементов. Задать начальные значения элементов массива прямо при объявлении массива и вывести элементы массива в виде колонки чисел в обратном порядке с указанием их порядкового номера элемента массива.. Для вывода использовать оператор цикла while и функцию cout.
15.Создать массив целых чисел из 7 элементов. Задать начальные значения элементов массива прямо при объявлении массива и вывести каждый второй элемент массива в виде колонки чисел в обратном порядке. Для вывода использовать оператор цикла for и функцию printf.
16.Создать массив вещественных (дробных) чисел из 7 элементов. Ввести значения элементов массивов с клавиатуры при помощи функции cin и вывести каждый второй элементы массива в виде колонки чисел в обратном порядке. Для вывода использовать оператор цикла while и функцию cout.

Задание №2. На основании задания к лабораторной работе №1 разработать программу работы с массивом.
Например, рассмотрим вариант 1. Разработать программу определения типа самолета для перевозки каких-либо грузов. В программе вводится вес груза и на его основании осуществляется выбор типа самолета. Груз менее 5т способен перевозит "ТУ-134". Вес более 5т, но менее 30т - "ИЛ-86". Вес более 30 т, но менее 45 "Руслан".
Модифицированное задание: Необходимо, разработать программу определения типа самолета для перевозки нескольких грузов. Данные о грузах хранятся в массиве. При выводе результатов выводится вес груза и тип самолета, который необходим для перевозки груза. Груз менее 5т способен перевозить "ТУ-134". Вес более 5т, но менее 30т - "ИЛ-86". Вес более 30т, но менее 45т - "Руслан".

Лабораторная работа №3
Функции,
поиск минимального и максимального значения массива

1.Поиск минимального и максимального элементов массива

Процесс поиска минимального или максимального элемента массива обычно осуществляется при помощи операторов цикла. Перед запуском цикла

предполагается, что один из элементов массива является минимальным или максимальным (обычно это первый элемент массива). Значение выбранного элемента массива принимается за эталонное. Цикл выполняется для всех элементов массива за исключением выбранного. Если выбран первый элемент массива, то цикл можно организовывать со второго элемента массива. В цикле осуществляется сравнение значений элементов массива с выбранным. Если при этом сравнении один из элементов массива оказывается больше (поиск максимального значения) или меньше (поиск минимального значения) текущего элемента, то значение текущего элемента массива принимается за эталонное. Цикл продолжает свое выполнение до нахождения нового эталонного значения или до его остановки (после перебора всех значений массива).

Рассмотрим пример. Необходимо найти минимальное значение элементов массива из 5 целых чисел. Алгоритм действий следующий: объявим массив целых чисел "a" из 5 элементов и задаем (или вводим с клавиатуры или файла) значения элементов массива, сохраняем количество элементов в массиве в переменной "n". Принимаем, что первый элемент массива является минимальным. Организуем цикл перебора элементов массива, начиная со второго элемента массива. В цикле осуществляется проверка - является ли значение, сохраненное в min, меньше значения текущего элемента. Если значение текущего элемента оказывается меньше, чем значение, находящееся в min, то заносим его в переменную min. После выполнения описанных действий в переменной min будет находиться минимальное значение массива.

```
int a[5]={3,5,4,1,2};      //  создан массив целых чисел из 5 элементов
n=5;                       //  определение количества обрабатываемых элементов
min=a[0];                  //  принимаем, что первый элемент массива
                           //  является минимальным
for(i=1;i<n;i++)           //  цикл перебора элементов массива
    if(min>a[i])           //  проверка, является ли значение, сохраненное в min,
                           //  меньше значения текущего элемента
        min=a[i];          //  если значение текущего элемента
                           //  оказывается меньше min
```

Рассмотрим более подробно пример выполнения программы при поиске минимального числа среди последовательности чисел 3,5,4,1,2. При старте цикла в min=3, т.к. a[0]=3. Цикл выполнится 4 раза, т.е. переменная i будет принимать значения 1,2,3,4 (это задано в операторе цикла for). При i=1 в цикле сравнивается значения min=3 и a[1]=5. Значение min меньше значения a[1], т.е. условие min>a[1] не выполняется. При i=2, сравниваются значения min=3 и a[2]=4. Значение min меньше значения a[2], т.е. условие min>a[2] не выполняется.

При i=3 сравниваются значения min=3 и a[3]=1. Значение min больше значения a[3], т.е. найдено значение, меньше min. Значение min=a[3]. Это значение используется как эталонное для дальнейшего поиска минимального (до тех пор будет найдено новое наименьшее значение). При i=4 сравниваются значения min=1 и a[4]=4. Если значение в min меньше, то условие min>a[1] не выполняется.

При необходимости нахождения максимального элемента массива в операторе сравнения меняется знак с "больше" на "меньше".

2.Функции

Процесс разработки программного обеспечения предполагает расчленение сложной задачи на набор более простых подзадач. В Языке Си поддерживаются функции как логические единицы (поименованные блоки текста программы), служащие для реализации отдельных подзадач.

Функции в Си должны иметь уникальные имена. Существенное значение имеет тип возвращаемого значения функцией. Функция может возвращать только одно значение. Функция может иметь параметры, необходимые для ее выполнения. Список параметров объявляется в круглых скобках после имени функции. Количество параметров может быть произвольным.

Простейшим примером использования функций является функция main (головная), которая должна присутствовать в любой программе, разрабатываемой на Си. С нее начинается выполнение программы. Чаще всего функция main не имеет принимаемых параметров и не возвращает значения. В этом случае вместо отсутствующих типов указывается ключевое слово void (если тип возвращаемого параметра не указан, то по умолчанию тип возвращаемого значения такой функции является int). Пример: void main(void).

Если функция возвращает значение, то это значение передается вызвавшему фрагменту при помощи записываемого в операторе «return» выражения. Return вызывает завершение работы данной функции, передачу значения в функцию, вызвавшую данную, и возврат управления на оператор, следующий после вызова функции.

При вызове функции указываются фактические параметры вызова, их количество должно соответствовать числу и типу формальных параметров в заголовке вызываемой функции. Если функция не возвращает значения (т.е. возвращает void), то оператор return может использоваться в варианте «return;» (без следующего за ним выражения). Пример:

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
```

```
float pi(void)
{                                   // функция только возвращает
                                    //значение константы ПИ
    return  3.14159265359;          // возвращаемое значение
}
int SummaAandB(int A,int B)
{                                  //  функция возвращает сумму переменных A и B
     return A+B;                   // возвращаемое значение
}
void PrintName(void){     //функция выводит на экран имя Serg 10 раз по одному
                          //в строке, не возвращает и не принимает никаких значений
  for(int t=0;t<10;t++)
      printf("Serg\n");
}
void PrintA(int A){ //  функция выводит на экран значение переменной A
   printf("%d",A);
}
void main(void){
   float f=pi();                    // вызов функции pi
   printf("pi=%f\n",f);
   int i=SummaAandB(1,3);           // вызов функции SummaAandB
   printf("summa(A+B)=%d\n",i);
   PrintName();                     // вызов функции PrintName
   PrintA(20);                      // вызов функции PrintA
}
```

Варианты задания (все логически законченные части программы оформить в виде отдельных функций):

1.Разработать программу для расчета отношения величины средней зарплаты сотрудников фирмы к максимальной зарплате. Расчет средней зарплаты осуществляется для сотрудников, зарплата которых выше 100 дол.

2.Разработать программу для расчета отношения величины средней зарплаты сотрудников фирмы к минимальной зарплате. Расчет средней зарплаты осуществляется для сотрудников, зарплата которых выше 25 дол.

3.Разработать программу выбора типа самолета для перевозки группы грузов. При выборе типа самолета исходить из того, что грузы являются неделимыми. Для перевозки может использоваться только один самолет. Груз менее 5 т способен перевозить "ТУ-134". Вес более 5 т, но менее 30 т - "ИЛ-86". Вес более 30 т, но менее 45 т - "Руслан".

4.Разработать программу тестирования партий изделий. При тестировании определяется процент брака. Определить номера 3-х партий изделий с наибольшим процентом брака и вывести их средний процент.

5.Разработать программу определения. При тестировании определяется процент брака изделий. Определить номера 3-х партий изделий с наибольшим процентом брака и вывести их средний процент брака.

6.Разработать программу определения призовых мест на соревнованиях по прыжкам в длину. Определить номера 3-х призовых мест на основании информации о длине прыжка. Вывести наименьшую длину прыжка.

7.Разработать программу определения номера прыжка в длину максимальной длины (позиции максимального элемента в массиве). Определить номера 3-х призовых мест на основании информации о длине прыжка. Вывести наименьшую длину прыжка (кроме переменной max, в которой сохраняется максимальное значение, необходимо ввести переменную, где будет сохраняться позиция этого элемента в массиве).

8.Разработать программу определения номера прыжка в длину минимальной длины (позиции минимального элемента в массиве). Определить номера 3-х призовых мест на основании информации о длине прыжка. Вывести наименьшую длину прыжка. (кроме переменной min, в которой сохраняется минимальное значение, необходимо ввести переменную, где будет сохранять позиция этого элемента в массиве).

9.Разработать программу автоматического контроля качества знаний у студентов при выполнении теста. При количестве ошибок менее 2-х ставится оценка 5, более 1-й и менее 4-х ошибок - оценка 4, более 3 ошибок - оценка 3. Определить минимальное и максимальное количество ошибок.

10.Разработать программу нахождения максимального среднего балла у студентов, считается, что средний балл после 3-х экзаменов определяется при вводе информации (используется только одномерный массив для хранения среднего балла).

11.Разработать программу определения среднестатистического количества осадков в течение года. Количество осадков вводится по месяцам. Определить отношение среднестатистического количества осадков к максимальному количеству.

12.Разработать программу определения отношения минимального количества осадков в течение года к максимальному количеству. Количество осадков вводится по месяцам.

13.Разработать программу поиска среднестатистического значения среди значений осадков за год. Количество осадков вводится по месяцам.

14.Разработать программу отбора пилотов- испытателей. Отбор осуществляется на основе общего количества удачных испытаний. Найти 2 лучших.

15.Разработать программу определения диапазона цен на данный товар в различных магазинах. Анализ осуществляется на основе цен в определенных магазинах.

16.Разработать программу определения отношения минимальной среднемесячной температуры в течение года к максимальной температуре.

Лабораторная работа №4
Указатели. Связь массивов и указателей

1.Указатели

Указатель - это адрес некоторого объекта, через него можно обращаться к этому объекту. Унарная операция & дает адрес переменной. Операцию & можно применять только к переменным и элементам массива. Пример присвоения адреса переменной "x" переменной-указателю "y":

```
int x,*y;
y=&x;
```

Унарная операция * воспринимает свой операнд как адрес некоторого объекта и использует этот адрес для выборки содержимого. Пример извлечения значения по адресу указателя

```
int x,*y,z;
y=&x;
z=*y;
```

Указатели можно использовать как операнды в арифметических операциях. Если y- указатель, то унарная операция y++ увеличивает его значение. Для y++ - адрес следующего элемента. Указатели и целые числа можно суммировать. Конструкция y+n (y - указатель, n - целое число) задает адрес n-го объекта, на который указывает y. Это справедливо для любых объектов (int, char, float и т.п.). Транслятор будет масштабировать приращение адреса в соответствии с типом, определенным в соответствии с объявлением (int *y; char *y; float *y).

Любой указатель можно проверить на равенство (==) или неравенство (!=) со специальным значением NULL, которое записывается вместо нуля. Слово NULL позволяет определить указатель, который ничего не адресует.

```
int *y;
y=NULL;
int x=5;
if(y==NULL) y=*x;
```

Обратите внимание, при создании указателя в нем находится произвольное значение, т.е. он ссылается на любой фрагмент памяти. Если вы по тексту программы проверяете значение указателя на NULL, то после объявления указателя его необходимо приводить к значению NULL.

2.Связь массивов и указателей

Если объявить

int mas[100], *p, a;

то:

1) для массива отводится память в адресном пространстве под 100 элементов типа int;

2) память отводится под указатель-константу с именем MAS, значением указателя является адрес массива;

3) память отводится под указатель-переменную с именем p.

Операция инициализации указателя может осуществляться только операцией "присвоить адрес некоторой переменной".

p = &a;

p = &mas[0]; или p = mas;

или присвоением p = NULL;, где NULL - константа, определенная через define в файле NULL.H.

Допустимо p=0, но не рекомендуется.

Ошибкой являются:

a=10;

p=a; // где p - указатель. Присвоение невозможно, так как типы int* и int.

p=10; // присвоение невозможно, так как типы int* и const int.

Указателю нельзя присваивать целые значения, но можно складывать и вычитать указатель и целые числа.

p+=10; - экв. p = p+10; - увеличение адреса на 10* масштабный множитель.

p-=2; уменьшение на 2* масштаб множителя.

p+=10; увеличивает на 10 содержимое ячейки, на которую ссылается p.

Например:

Если p=mas; то p+=10; эквивалентно p=p+10 и эквивалентно присвоению p=&mas[10];

*p+=10; эквивалентно mas[0]=mas[0]+10;

Если 2 указателя ссылаются на элементы одного и того же массива, то допускаются операции отношения над ними: = =; !=; <, >, и т.д.

Для указателей, ссылающихся на элементы разных массивов, результат сравнения не определен.

Допускается вычитание указателей.

Например, разработаем функцию вычисления длины строки:

```
int strlen(char *s)
{       char *p = s;  // объявлен указатель и инициирован адресом
                      // массива символов.
        while(*p! = '\0')       p++;
        return p-s;
}
```

Данная функция возвращает значение типа int, т.е. длина строки не может превышать значения, которое представимо типом int (2-байтным целым). Поэтому в реальных программах лучше пользоваться стандартной функцией strlen, которая описана в файле string.h. Эта функция умеет выбирать тип возврата в зависимости от модели памяти, используемой в программе. Для этого определен (define) тип ptrdiff_t в файле STDDEF.H.

Задание: в соответствии с вариантами к лабораторной работе №3 разработать программу с использованием указателей.

Лабораторная работа №5
Многомерные массивы. Матрицы.

Массивы в С могут иметь много индексов. Обычным представлением многомерных массивов являются таблицы значений, содержащие информацию в строках и столбцах. Чтобы определить отдельный табличный элемент, нужно указать два индекса: первый указывает номер строки, а второй – номер столбца. Таблицы или массивы, которые требуют двух индексов для указания отдельного элемента, называются двумерными массивами. Многомерный массив может иметь более двух индексов. Каждый элемент в массиве a определяется именем элемента в форме a[i][j]; a – это имя массива, a i и j – индексы, которые однозначно определяют каждый элемент в a. Имена элементов первой строки имеют первый индекс 0.Многомерные массивы могут получать начальные значения в своих объявлениях точно так же, как массивы с единственным индексом.
Пример:

```
int a[2][2]={{1,2},{3,4}};
```

Матрица - это двумерный массив, т.е. количество измерений массива определяется при декларации по количеству квадратных скобок после имени массива. Пример:

```
char a[10][30] ;
int b[10][30];
float c[10][30];
```

Двумерный массив представляется как одномерный, элементы которого тоже массивы. Элементы двумерного массива хранятся по строкам, т.е. если проходить по ним в порядке их расположения в памяти, то быстрее всего изменяется самый правый индекс. Например, обращение к девятому элементу пятой строки запишется так: a[4][8].

Матрицы бывают квадратными и прямоугольными. У квадратных матриц количество строк и количество элементов в строке одинаковое.

Варианты задания:

1.Дана квадратная матрица M(4x4). Ввести данные в матрицу с клавиатуры. Посчитать сумму диагонали матрицы.

2.Дана квадратная матрица M(4x4). Ввести данные в матрицу с клавиатуры. Посчитать сумму обратной диагонали матрицы.

3.Дана квадратная матрица M(4x4). Ввести данные в матрицу с клавиатуры. Посчитать сумму всех элементов матрицы.

4.Дана квадратная матрица M(4x4). Ввести данные в матрицу с клавиатуры. Посчитать сумму столбцов матрицы и занести ее в массив A, состоящий из 4 элементов.

5.Дана квадратная матрица M(4x4). Ввести данные в матрицу с клавиатуры. Посчитать сумму строк матрицы и занести ее в массив A, состоящий из 4 элементов.

6. Дана матрица M(6x4). Ввести данные в матрицу с клавиатуры. Поменять местами четные и нечетные столбцы матрицы.

7. Дана матрица M(4x6). Ввести данные в матрицу с клавиатуры. Поменять местами четные и нечетные строки матрицы.

8. Дана матрица M(4x6). Ввести данные в матрицу с клавиатуры. Перевернуть матрицу по горизонтали.

9. Дана матрица M(4x6). Ввести данные в матрицу с клавиатуры. Перевернуть матрицу по вертикали.

10. Дана матрица M(4x4). Ввести данные в матрицу с клавиатуры. Перевернуть матрицу относительно ее диагонали.

11. Дана матрица M(4x4). Ввести данные в матрицу с клавиатуры. Перевернуть матрицу относительно ее обратной диагонали.

12. Дана матрица M(4x4). Ввести данные в матрицу с клавиатуры. Определить четверть с наибольшей суммой элементов.

13. Дана матрица M(4x4). Ввести данные в матрицу с клавиатуры. Перевернуть четные ряды матрицы.

14. Дана матрица M(4x4). Ввести данные в матрицу с клавиатуры. Перевернуть нечетные строки матрицы.

15. Дана матрица M(4x6). Ввести данные в матрицу с клавиатуры. Найти сумму максимальных элементов строк матрицы.
16. Дана матрица M(4x6). Ввести данные в матрицу с клавиатуры. Найти сумму максимальных элементов столбцов матрицы.

Варианты дополнительного задания:
1.В сессию студенты одной группы сдали по 3 экзамена. Результаты сессии представлены в виде матрицы M(5x3). Ввести данные в матрицу с клавиатуры. Определить средний балл по каждому предмету. Результаты занести в одномерный массив N, содержащий 3 элемента, и вывести на экран.
2.В сессию студенты одной группы сдали по 3 экзамена. Результаты сессии представлены в виде матрицы M(5x3). Задать значения элементов матрицы при ее объявлении. Определить средний балл по каждому предмету. Результаты занести в одномерный массив N, содержащий 3 элемента, и вывести на экран.
3.В сессию студенты одной группы сдали по 3 экзамена. Результаты сессии представлены в виде матрицы M(5x3). Ввести данные в матрицу с клавиатуры. Определить общее количество пятерок, четверок, троек у данной группы. Результаты занести в одномерный массив N, содержащий 3 элемента, и вывести на экран.
4.В сессию студенты одной группы сдали по 3 экзамена. Результаты сессии представлены в виде матрицы M(5x3). Задать значения элементов матрицы при ее объявлении. Определить размер стипендии для всей группы, исходя из следующих данных: за каждую пятерку студент получает 2 условных рубля, за четверку 1 условный рубль, а за тройку студент ничего не получает.

Лабораторная работа №6
Структурированные типы данных

1.Структуры

Структура – это объединение одного или более объектов (переменных, массивов, указателей, других структур и т.п.). Как и массив она представляет собой совокупность данных. Отличием является то, что к ее элементам необходимо обращаться по имени и что различные элементы структуры необязательно должны принадлежать одному типу. Объявление структуры осуществляется с помощью ключевого слова struct, за которым идет ее тип и далее список элементов, заключенных в фигурные скобки:

```
struct тип {
```

```
        тип_элемента_1 имя_элемента_1;
…
        тип_элемента_n имя_элемента_n;
};
```

Именем элемента может быть любой идентификатор. Как и выше, в одной строке можно записывать через запятую несколько идентификаторов одного типа. Рассмотрим пример:

```
struct date {
    int day;
    int month;
    int year;
};
```

Следом за фигурной скобкой, заканчивающей список элементов, могут записываться переменные данного типа, например: struct date {int day;int month; int year;} a,b,c; В этом случае для каждой переменной выделяется память, объем которой определяется суммой длин всех элементов структуры. Описание структуры без последующего списка не вызывает выделения никакой памяти, а просто задает шаблон нового типа данных, которые имеют форму структуры. Введенное имя типа позже можно использовать для объявления структуры, например struct date days;. Теперь переменная days имеет тип date. При необходимости структуры можно инициализировать, помещая за объявлением список начальных значений элементов. Разрешается вкладывать структуры одна в другую, например:

```
struct man { char name [30], fam [20];
                struct date bd;
                int voz; };
```

Определенный выше тип data включает три элемента: day, month, year, содержащий целые значения (int). Структура man включает элементы: name[30], fam[20], bd и voz. Первые два name[30] и fam[20] – это символьные массивы из 30 и 20 элементов каждый. Переменная bd представлена составным элементом (вложенной структурой) типа data. Элемент voz содержит значения целого типа (int). Теперь разрешается объявить переменные, значения которых принадлежат введенному типу:

```
struct man _ man_ [100];
```

Здесь определен массив _man_, состоящий из 100 структур типа man. В языке Си разрешено использовать массивы структур. Структуры могут состоять из массивов и других структур.

Чтобы обратиться к отдельному элементу структуры, необходимо указать ее имя, поставить точку и сразу за ней написать имя нужного элемента, например:

```
_man_ [i].voz = 16;
```

man [j].bd.day = 22;

man [j].bd.year =1976;

При работе со структурами необходимо помнить, что тип элемента определяется соответствующей строкой объявления в фигурных скобках. Например, _man_ имеет тип man, year - является целым числом и т.п. Поскольку каждый элемент структуры относится к определенному типу, его имя может появляться везде, где разрешено использовать значения этого типа. Допускаются конструкции вида _man_[i] = _ man_[j]; где _man_[i] и _man_[j] - объекты, соответствующие единому описанию структуры. Другими словами, разрешается присваивать одну структуру другой по их именам.

Унарная операция & позволяет взять адрес структуры. Предположим, что задано объявление

struct date { int d,m,y; } day;

Здесь day - это структура типа date, включающая три элемента: d,m,y. Другое объявление struct date _*db; устанавливает тот факт, что db - это указатель на структуру типа date. Запишем выражение: db = &day;. Теперь для выбора элементов d, m, y структуры необходимо использовать конструкции: (*db).d, (*db).m, (*db).y. Действительно, db - это адрес структуры, *db - сама структура. Круглые скобки здесь необходимы, так как точка имеет более высокий, чем звездочка приоритет. Для аналогичных целей в языке Си предусмотрена специальная операция ->. Она тоже выбирает элемент структуры и позволяет представить рассмотренные выше конструкции в более простом виде: db->d, db->m, db->y.

2.Битовые поля

Особую разновидность структур представляют поля. Поле - это последовательность соседних бит внутри одного целого значения. Оно может иметь тип signed int либо unsigned int и занимать от 1 до 16 бит. Поля размещаются в машинном слове в направлении от младших к старшим разрядам.

Например, структура

struct prim {int a:2; unsigned b:3; int :5;

int c:1; unsigned d:5; } f,j;

обеспечивает размещение, показанное на рис. 4.6. Если бы последнее поле было задано так: unsigned d:6;, то оно размещалось бы не в первом слове, а в разрядах 0-5 второго слова.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
d	d	d	d	d	b	не используется					b	b	b	a	a

Рис. 4.6.Пример блока памяти, выделенного под битовое поле

В полях типа signed крайний левый бит является знаковым. Например, такое поле шириной 1 бит может только хранить значения -1 и 0, так как любая ненулевая величина будет интерпретироваться как -1.

Поля используются для упаковки значений нескольких переменных в одно машинное слово с целью экономии памяти. Они не могут быть массивами и не имеют адресов, поэтому к ним нельзя применять унарную операцию &. Пример:

```
#include <stdio.h>
struct  {unsigned a : 1;    unsigned b : 1;      unsigned y : 1;      unsigned c : 2; } f;
void main () {     int i;
    printf("размер f=%d байт \n",sizeof(f));
    f.a =f.b=1;    // в поля a и b записывается 1
    for (i=0;i<2;i++)
    {    f.y =f.a && f.b; //  конъюнкция a и b
         printf(" цикл %d;f.y =%d\n", i, f.y);
         f.b=0;     }
    f.c = f.a +!f.b;              // сложение значений f.a и отрицание f.b (=1)8)
printf("f.c=%d", f.c);}       // f.c=2
```

<u>результаты работы:</u>

размер f=1 байта
цикл o;f.y =1
цикл 1;f.y =0
f.c=2

3.Смеси

Смесь - это разновидность структуры, которая может хранить (в разное время) объекты различного типа и размера . В результате появляется возможность работы в одной и той же области памяти с данными различного вида. Для описания смеси используется ключевое слово union, а соответствующий синтаксис аналогичен синтаксису структуры.

Пусть задано объявление:

union r { int ir; float fr; char cr; } z;

Здесь ir имеет размер 2 байта, fr - 4 байта и cr - 1 байт. Для z будет выделена память, достаточная чтобы сохранять самый большой из трех приведенных типов. Таким образом, размер z будет 4 байта. В один и тот же момент времени в z может иметь значение только одна из указанных переменных (ir, fr, cr). Пример:

```
#include <stdio.h>
union r{ int ir; float fr; char cr;} z;
```

```
float f;
// объявлена смесь z типа r: .Размер смеси будет определяться  размером самого
//длинного элемента, в данном случае fr,
void main (void)
{       //в версии Borland C++ версии 3.1 обнаружена  ошибка при
        //использовании в вычислениях и преобразованиях вывода
        //вещественных значений элементов структур . Чтобы обойти ошибку,
        //выбираем вещественное значение элемента union  в простую
        //вещественную переменную f (f=z.fr;), а затем используем f в
        //выражениях и наоборот.
printf ("размер z=%d байта \n",sizeof(z));
        // sizeof(z) вычисляет длину переменной z и printf  распечатывает
        //вычисленную длину
printf ("введите значение z.ir \n");     //выдача приглашения для ввода
scanf  ("%d",&z.ir);                     //ввод целого значения в элемент z.ir
printf ("значение ir =%d \n",z.ir);      //вывод значения z.ir
printf ("введите значение z.fr \n");     //приглашение для ввода
                                          //вещественного значения
scanf  ("%f",&f);             //ввод вещественного значения в переменную f и
z.fr=f;          //запись в z.fr (фактически реализован ввод: scanf  ("%f",&z.ir);.
printf ("значение fr=%f \n",f);    //вывод значения вещественной переменной
printf ("введите значение z.cr \n");          // приглашение на ввод информации
flushall();                       // очистка буферов ввода-вывода.
        //Такая очистка буфера здесь необходима, так как в буфере ввода остается
        //символ конца строки от предыдущего ввода, который затем введется
        //спецификацией %c , вместо реально набираемого символа
scanf  ("%c",&z.cr);                      //чтение символа, введенного с клавиатуры
printf ("значение cr=%c;\n",z.cr);        //вывод значения символа
}
```

Пример сеанса работы с программой («Enter» - это нажатие этой клавиши):

размер z= 4 байта
введите значение z.ir
7 «Enter»
значение ir=7
Введите значение z.fr
38.345678«Enter»
Значение fr=8.345678
Введите значение z/cr
P«Enter»

Значение cr= P;

Варианты задания (в программе использовать структуры, битовые поля и смеси; все логически законченные части программы оформить в виде отдельных функций):

1.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.

2. 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.

3. 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.

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

5. 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.

6. 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.

7. 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.

8. 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

9. 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.

10. 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.

11. 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.
12. 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.
13. 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.
14. 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.
15. Разработать программу оценки 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.
16. 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.

Лабораторная работа №7
Файлы

1.Файлы

Файл – это организованный набор данных , расположенных на внешнем носителе.

В файлах размещаются данные, предназначенные для длительного хранения. Каждому файлу присваивается используемое при обращении к нему уникальное имя.

В языке С отсутствуют инструкции для работы с файлами. Все необходимые действия выполняются через функции, включенные в стандартную библиотеку. Они позволяют работать с различными устройствами, такими, как диски, принтер, коммуникационные каналы и т.п. Эти устройства сильно отличаются друг от друга. Однако файловая система позволяет преобразовывать их в единое абстрактное

логическое устройство, называемое потоком. Существует два типа потоков: текстовые и двоичные.

Прежде чем читать или записывать информацию в файл, он должен быть открыт. Это можно сделать с помощью библиотечной функции fopen. Она берет внешнее представление файла (например C:\MY_FILE.TXT) и связывает его с внутренним логическим именем, которое используется далее в программах. Логическое имя – это указатель на требуемый файл. Его необходимо объявлять, и делается это, например, так:

```
FILE *lst;
```

Здесь FILE - имя типа, описанное в стандартном определении stdlo.h, 1st - указатель на файл. Обращение к функции fopen в программе производится так:

lst=fopen(спецификация файла, вид использования файла);

Спецификация файла может быть, например : C:\MY_FILE.TXT - для файла MY_FILE.TXT на диске C:; A:\MY_DIR\EX2_3.CPP - для файла EX2_3.CPP в поддиректории A:\MY_DIR и т.п. Вид использования файла может быть:

r - открыть существующий файл для чтения;

w - создать новый файл для записи (если файл с указанным именем существует, то он будет переписан)

a - дополнить файл (открыть существующий файл для записи информации, начиная с конца файла, либо создать файл, если он не существует);

pb - открыть двоичный файл для чтения;

wb - создать двоичный файл для записи;

ab - дополнить двоичный файл;

rt - открыть текстовой файл для чтения;

wt - создать текстовой файл для записи;

at - дополнить текстовой файл;

r+ - открыть существующий файл для записи и чтения;

w+ - создать новый файл для записи и чтения;

a+ - дополнить или создать файл с возможностью записи и чтения;

r+b - открыть двоичный файл для записи и чтения;

w+b - создать двоичный файл для записи и чтения;

a+b - дополнить двоичный файл с предоставлением возможности записи и чтения.

Если режим t или b не задан (например, r, w или a), то он определяется значением глобальной переменной _fmode. Если _fmode = 0_BINARY, то файлы открываются в двоичном режиме, а если _fmode =O_TEXT - в текстовом режиме. Константы 0_BINARY и O_TEXT определены в файле fcntl.h.

Строки вида r+B можно записывать и в другой форме: rb+. Если в результате обращения к функции fopen возникает ошибка, то она возвращает указатель на

константу NULL. После окончания работы с файлом, он должен быть закрыт. Это делается с помощью библиотечной функции fclose. Она имеет следующий прототип:
int fclose(FILE *lst);

При успешном завершении функция fclose возвращает значение нуль. Любое другое значение говорит об ошибке.

2.Вывод информации в файл

```
#include <stdio.h>

void main (void)
{
        char str[50];
        FILE  *rstr, *wstr, *pstr, *astr;
        rstr = fopen ("c:\\ my_file.txt", "rt");
        wstr = fopen ("c:\\out_file.txt", "wt");
        pstr = fopen ("prn", "wt");
        astr = fopen ("c:\\out_plus.txt", "at");
        while (fscanf (rstr, " %s ", str) !=EOF)
        {
            printf ( " Вывод на дисплей: %S\n", str);
            fprintf (wstr, "%s\n", str);/*запись файла (прежнее содержание стирается)*/
            fprintf (pstr, "%s\n", str); /* вывод на печать*/
            fprintf (astr, "%s\n", str); /*дополнение файла*/
        }
        fclose(rstr); fclose(wstr); fclose(pstr); fclose (astr);
}
```

В данном примере указатели не инициализируются адресами соответствующих файлов, открытых для указанного типа операций.
Имя “prn”, используемое для вывода на печать, представляет собой стандартное имя устройства печати.

3.Чтение строк из файла и вывод их на экран

```
#include <stdio.h>
void main (void)
```

```
{
        char     str [50];
        FILE * fr, * fw;
        if ((fr=fopen ("A:\\fail.ttt","r+" ))==NULL)
        //открытие файла с дискеты
        {
                printf("Файл не открылся. \nВведите информацию с клавиатуры");
                fgets (str ,49, stdin); // можно gets (str ,49);
        }
        else
        fgets (          str ,49, fr);              // или введите строку      до 49
        printf ("Вывод строки: %s", str);        // символов без пробела
        if ((fw=fopen ("a:\\1.txt", "w+"))==NULL)
        {    printf("Файл не открылся");  }
        else
        {    printf("\n в файл 1.txt "); //запись в файл
            fputs (str, fw);              // функция записывает
                return;
         } //выход из программы
        // если файл не открылся, то вывод из str в файл ошибок.
        fprintf (stderr, " Вывод в стандартный файл для ошибок\n%s",str);
        fclose (fr); fclose (fw);
}
```

Программа считывает из файла fail.ttt дискеты, вставленной в дисковод A: 49 символов или пока не встретится символ конец строки. Если файл не открылся, то предлагает ввести информацию с клавиатуры (введется 48 символов или до нажатия клавиши Ввод). Затем информация выводится в файл 1.txt на дискете или, если не удалось его открыть, в файл ошибок на экран.

Задание: Использовать варианты задания к лабораторной работе №6. Модифицировать программу, чтобы данные могли сохраняться в файле и считываться из него.

Лабораторная работа №8
Динамические переменные, работа с памятью

1.Операторы new и delete

Операторы new и delete обычно используются вместе. Оператор new выделяет память под переменную, а delete ее освобождает. Синтаксис данных операторов:

имя_указателя=new (имя_переменной);

delete (имя_укзателя);

Пример:

```
int  *n;          // объявление указателя
n = new int;      // выделение памяти
delete (n);       // освобождение памяти
```

Часто операторы new и delete используются для работы с массивами, длина которых заранее неизвестна. Например, количество элементов массива определяется в программе непосредственно перед использованием массива. В этом случае синтаксис оператора new следующий:

имя_указателя = new тип_элементов_массива (количестов_элемнтов);

Пример – необходимо посчитать сумму элементов массива переменной длины. 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;
        printf("Введите количество элементов массива:\n");
        scanf("%d",&k);
        a = new (int[k]);              // выделение памяти под массив
        for(int l=0;l<k;l++)           // ввод элементов массива
              scanf("%d",&a[l]);
        int sum=0;
        for(int l=0;l<k;l++)           // расчет суммы элементов массива
              sum=sum+a[l];
        printf("%d",sum);              // вывод суммы элементов массива
        delete(a);                     // освобождение памяти
}
```

2.Функции malloc и free

Операторы malloc и free обычно используются вместе. Оператор malloc выделяет необходимое число байт в памяти под переменную, а free ее освобождает. Синтаксис вызова данных функций:

имя_указателя= (тип_указателя) malloc (количество_выделяемых_байтов);

free(имя_укзателя);

Пример:

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

Пример выделения памяти под строку символов:

```
include <string.h>
#include <stdio.h>
#include <alloc.h>
void main(void)
{
  char *str;
  str = (char *) malloc(10);       // выделение памяти под строку символов
  strcpy(str, "Hello");            // присваивание значения
  printf("String is %s\n", str);  // вывод на экран
  free(str);                       // освобождение памяти
}
```

Проверка на ошибки при выделении памяти:

```
if ((str = (char *) malloc(10)) == NULL)
{
  printf("Ошибка\n");      //  сообщение о ошибке
  exit(1);                 // выход
}
```

Задание: Использовать варианты задания к лабораторной работе №6. Модифицировать программу, чтобы данные хранились в массиве переменной длины. Длина массива должна определяться при запуске программы.

Литература

1. Фигурнов В.Э. Программное обеспечение персональных ЭВМ. – М.: Наука, 1988.
2. Гукин Д. Word for Windows для начинающих: Пер. с англ. – Киев.: Диалектика, 1994.
3. Бемер С., Фратер Г. MS Access для пользователя: Пер. с нем. – Киев.: Торгово-издат. Бюро BHV, 1994.
4. Николь Наташа, Албрехт Ральф. Электронные таблицы Excel 5.0: Практич. пособие. - М.: ЭКОМ., 1994.
5. Нортон П. Программно- аппаратная организация IBM PC. Пер.с англ. -М. : Радио и связь, 1992.
6. Керниган Б. Ритчи Д. Язык программирования Си. – М.: Финансы и статистика, 1985.
7. Уэйт М., Прата С., Мартин Л, Язык Си. – М.: Мир, 1988.
8. Бруно Бабе. Просто и ясно о Borland C++: Пер. с англ. – М. Бином.,1988.
9. Касаткин А.И., Вальвачев А.Н. От TURBO C к Borland C++. Мн.: Выш. шк., 1992.

Приложение 1

Работа с (IDE) Borland C++.

Интегрированная среда программирования (IDE) Borland C++.

Интегрированная среда (IDE) – это программа, имеющая встроенный редактор текстов, подсистему работы с файлами, систему помощи, встроенный отладчик, подсистему управления компиляцией и редактированием связей, а также компилятор и редактор связей. Другими словами, IDE дает возможность получить EXE-файл, не используя другие программы. IDE запускается файлом BC.EXE.

После запуска на исполнение файла запуска IDE (файл BC.EXE) на экране отображается основное окно IDE (рис 1).

Верхняя строка окна – это главное меню. Опции меню позволяют обратиться к подменю и выбрать соответствующую команду.

Нижняя строка экрана отведена под строку состояния, где выделены назначения «горячих» клавиш, воспринимаемых на данном этапе работы.

Выбрать любую из команд меню можно одним из трех способов:

1)нажать клавишу F10 и с помощью клавиш со стрелками выбрать необходимую команду;

2)установить курсор мыши на любое ключевое слово меню и нажать левую кнопку мыши;

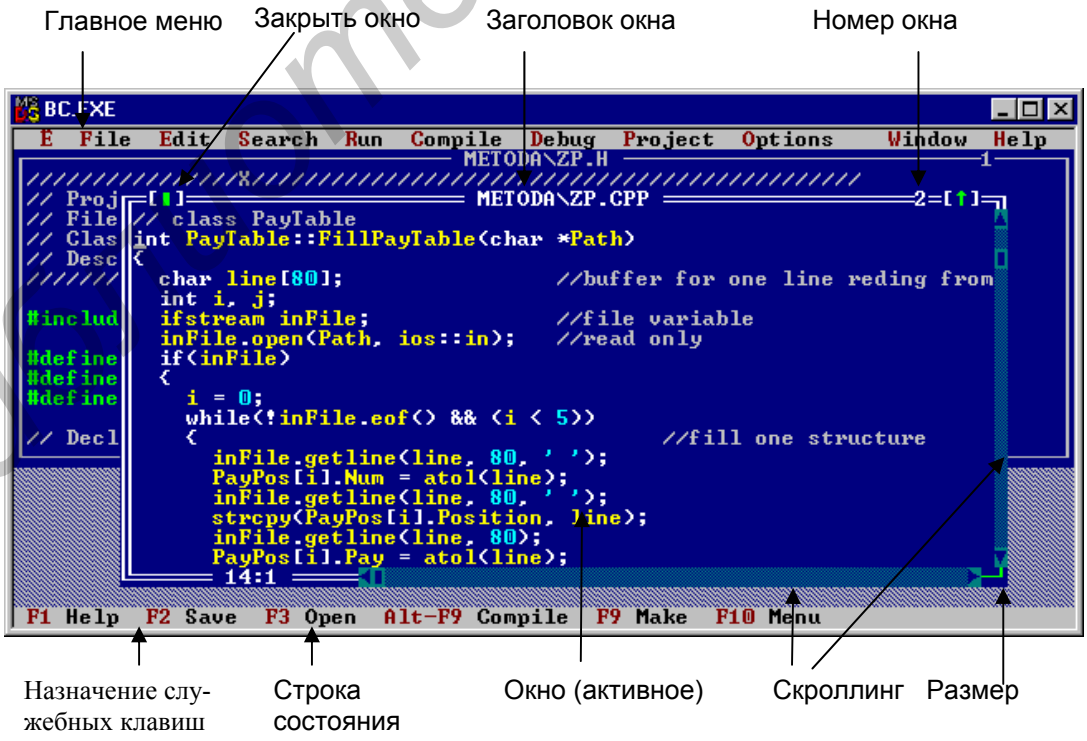

Рис. 1. Основное окно Borland C++ с двумя открытыми окнам

3)использовать «горячие» клавиши (метод скорейшего вызова команды). Одновременное нажатие клавиши Alt и «горячей» (клавиша подсвеченная другим цветом).

Окно – это ограниченная рамкой область экрана. Его можно открыть, переместить, покрыть другими окнами, изменить размеры, закрыть. Активное окно (то, которое воспринимает нажатия клавиш) обозначается двойной линией. Активное окно изображается всегда поверх других окон. В любой момент только одно окно может быть активным. Каждое окно имеет заголовок и номер, показанные в его верхней строке. Для активного окна там же расположены два управляющих поля, заключенных в квадратные скобки: поле справа используется для раскрытия окна мышью на полный экран, а поле слева – для закрытия окна. Многие окна для управления положением текста имеют по правой и нижней границам вертикальную и горизонтальную полосы прокрутки (скроллинга). Поля изменений размеров окна – это символы нижних углов рамки окна.

Операции с окнами могут выполняться тремя способами:
1)через команды главного меню Window;
2)с помощью манипулятора мышь;
3)при помощи «горячих» клавиш.

Закрыть можно только активное окно. Для этого либо выбирается в меню Windows команда Close, либо нажимается «горячая» клавиша Alt-F3, либо мышью выбирается поле [•] на окне.

Переключение между окнами выполняется так. Выбирается команд List… из меню Window или нажимается «горячая» клавиша Alt-0. Открывается окно диалога, в котором можно выбрать любое из открытых и ранее закрытых окон. Если на экране отображена хотя бы небольшая часть необходимого окна, достаточно в эту область установить мышь и нажать ее левую кнопку, чтобы окно стало активным. Циклический просмотр окон возможен при помощи клавиши F6.

Активное окно может быть раскрыто на весь экран либо выбором Window-Zoom, либо нажатием клавиши F5, либо выбором мышью поля [↑].
Для просмотра результатов выполнения программы, если вывод выполняется в текстовом режиме, используется переключение в окно вывода (Window - Output). Для просмотра результатов как в текстовом, так и графическом режимах, следует активизировать окно экрана пользователя (Window – User screen) или воспользоваться . одновременным нажатием клавиши – Alt-F5. Возврат в среду происходит при нажатии любой клавиши.

Переключение в режим редактирования выполняется автоматически при выборе команды New в меню File или при открытии файла. Для возврата из меню в режим редактирования достаточно нажать клавишу Esc.

Команды вставки и удаления (под блоком понимается выделенное подсветкой подмножество символов):

Ins – режим вставки/замены;

Del – удалить символ в позиции курсора;

Backspace – удалить символ слева от курсора;

Ctrl-Y – удалить строку;

Ctrl-N – вставить строку.

Команды работы с блоками:

Shift+клавиши со стрелками – выделение блока текста;

Ctrl-Ins – копировать блок в буффер обмена;

Shift-Ins – копировать блок из буффера обмена в текущую позицию курсора;

Ctrl-Del – удалить блок.

Shift -Del – вырезать блок в буфер обмена.

Этапы создания программы в инструментальных средах фирмы Borland.

Основные этапы создания программы в IDE Borland C++:

1)настройка опций среды программирования;

2)набор исходного текста программы;

3)компиляция программы;

4)компоновка программы;

5)отладка программы;

6)запуск программы на исполнение.

Система программирования Borland C++ включает:

1)интегрированную среду программирования (Integrated Development Environment - IDE);

2)компилятор исходного текста программы;

3)редактор связей (компоновщик);

4)библиотеки заголовочных файлов;

5)библиотеки функций;

6)программы-утилиты.

Задание опций интегрированной среды.

Первым шагом при работе с IDE является настройка нужных опций (дополнительных параметров). Все опции имеют значения по умолчанию. Рассмотрим основные опции, настраиваемые с помощью команд меню Options.

Для того чтобы начать работу с IDE, прежде всего требуется задать директории, используемые текстовым тедактором, компилятором и компоновщиком (рис. 2). Для этого используется команда Options\Directories (мы будем использовать формат записи Меню\Меню\…\Команда для экономии места). Поле ввода Include Directories

используется для задания директориев заголовочных файлов. В поле ввода разрешается указывать несколько директориев, разделяемых символом “;”.
Поле ввода Library Directories задает директории, содержащие объектный файл загрузчика (CO?.OBJ, где ? – это буква M, S, H, T, L, C в зависимости от используемой модели памяти) и файлы библиотек функций (.LIB). Поле ввода Output Directory задает директорий, в котором помещаются файлы с расширениями .OBJ, .EXE, .MAP. Если в поле – пустая строка, используется текущий директорий.

Рис. 2. Окно диалога для установки директориев.

При выборе строки Options\Compiler открывается еще одно меню для настройки опций компилятора. Наиболее важные опции задаются при выборе команды Code generation. Опция считается выбранной, если она помечена символом (•) и включенной, если она помечена символом [x]. Самым важным пунктом в окне Code generation является выбор модели памяти. Для большинства программ, разрабатываемых для ОС MS-DOS, нужно выбрать SMALL модель памяти.

Набор текста программы.

Следующим шагом является ввод программы с использованием текстового редактора и сохранение исходного текста программы в файле.
Набор текста программы можно выполнить встроенным или любым другим текстовым редактором. По традиции файлы, содержащие исходные тексты программ на языке C, имеют расширение имени файла .C, а на языке C++ - .CPP.

Не следует начинать компиляцию, компоновку или запуск программы без сохранения сделанного набора! Запущенная на выполнение программа может вызвать «зависание» компьютера, и сделанный набор будет потерян.
К программам-утилитам относят ассемблер, препроцессор, отладчик, программу профилирования и многие другие полезные программы.

Компиляция, редактирование связей, запуск программы на выполнение.

Borland C++ включает богатейшие библиотеки функций для управлениями файлами, выполнения ввода-вывода и многих других действий. Прототипы (заголовки функций с описанием типов формальных параметров и типа возвращаемого функцией значения), символические константы и другие макро, связанные с библиотечными функциями, объединяют в заголовочные файлы, которые по традиции имеют расширение .H. Необходимые для компиляции файлы включаются в текст программы при помощи препроцессорной директивы #include. При запуске на компиляцию текст программы сначала обрабатывается препроцессором, который обрабатывает только «свои» директивы (в частности, вместо директивы «#include имя_файла» встраивается из библиотеки INCLUDE файл, имя которого задано в директиве), а затем текст программы передается непосредственно на обработку компилятору.

Компиляция исходного текста программы инициируется либо через команду Compile\Compile to OBJ, либо нажатием «горячей» клавиши Alt-F9. Команда Make EXE file также запускает программу на компиляцию и при отсутствии синтаксических ошибок автоматически запускает компоновщик для получения .EXE-файла. Еще одна возможность для запуска программы на компиляцию – команда Run\Run (Ctrl-F9). После успешной компиляции и компоновке запускается полученный .EXE-файл на выполнение.

Все сообщения об ошибках и предупреждения IDE помещает в окно по имени Message. Это окно активно после завершения компиляции. Если в программе обнаружены ошибки, включаются средства трассировки ошибок, которые связывают строки текста программы в окне редактора со строками окна Message. Перемещение высвечивания клавишами со стрелками в окне Message синхронно сопровождается высвечиванием соответствующих ошибочных строк в тексте программы. При нажатии клавиши Enter активизируется окно редактора и курсор устанавливается на ошибочную строку. Нажатие клавиши F6 (переход или активизация следующего окна) вновь делает активным окно Message.

Многофайловая компиляция.

При модульном программировании не обойтись без многофайловай компиляции. При работе с большими программами намного удобнее размещать части программы не в одном, а в нескольких файлах. Каждый файл должен включать целиком одну или несколько функций. Имена этих файлов записываются в специальный файл – файл проекта, из которого IDE узнает, какие из текстовых файлов следует объединять в исполняемый (.EXE) файл.

Все необходимые для работы с файлами проектов команды включены в меню Project.

Для организации файла проекта необходимо открыть файл проекта. Для этого выполняется команда Project\Open Project… IDE активизирует специальное окно Project в нижней части экрана и открывает окно диалога, позволяющее загрузить нужный файл проекта или создать новый с заданным именем.

Если создан новый файл проекта, окно Project первоначально будет пустым. Включение файлов в проект и их удаление выполняются либо через команды Project\Add item… и Project\Delete item, либо нажатием клавиш Ins и Del, в случае если курсор размещен в окне Project. При добавлении файлов в проект открывается окно диалога, позволяющее выбрать нужный файл.

Окно Project упрощает переход от одного файла, включенного в проект, к другому при их редактировании. Для этого высвечивание перемещается на нужную строку окна Project и нажимается клавиша ENTER.

При работе со средой Borland C++ рекомендуется использовать проект, даже если программа состоит из одного файла.

Отладка программы

В процессе отладки вы можете:

1)осуществлять пошаговое выполнение программы. После прохода каждой ее строки будет производиться приостановка, позволяющая проанализировать промежуточные результаты;

2)проверять значение и местоположение (адрес) некоторой переменной в ходе выполнения программы;

3)просмотреть последовательность вызова функций в программе.

Существует два режима пошагового выполнения программы:

1)трассировка с заходом в тело функции, при встрече ее вызова в тексте программы (F7);

2)пошаговое выполнение функции (как обычной команды без захода в тело функции), при встрече ее вызова в тексте программы (F8).

Команда Run\Trace into (F7) запускает программу на отладку. Интегрированная среда высвечивает строку программы, содержащую точку входа main(). После этого нажатием клавиши F7 вызывается выполнение кода, соответствующего одной строке текста программы. Если в строке записана ссылка на функцию, начинается трассировка по тексту тела функции. При необходимости выполнения строки функции за один шаг, используется клавиша F8 (команда Run\Step over).

Для ускорения процесса отладки используется команда Run\Go to cursor (F4). Программа выполняется до строки, в которой в данный момент располагается текстовый курсор. Можно также задать режим выполнения до точки останова (через опцию подменю “Debug\Toggle breakpoint” или одновременным нажатием клавиш Ctrl

и F8, в дальнейшем будем использовать запись “Ctrl+F4”). При этом строка в точке останова подсвечивается обычно красным фоном. Снять установку точку останова можно повторным выполнением описанной команды, размещая курсор на подсвеченной строке останова.

Для наблюдения за изменением значений переменных в ходе выполнения программы используется подменю Debug\Watches\Add watch или “Ctrl+F7”. В появившемся окне Add Watch (вызов окна Add Watch можно также получить если нажать клавишу Ins, предварительно сделав активным окно Watch) необходимо ввести имя переменной, значение которой необходимо просмотреть и нажать Ввод. Указанная переменная размещается в окне Watch, создаваемом в нижней части экрана, и, в процессе отладки, через это окно можно наблюдать за изменением размещенных в нем переменных. Удалить переменную из окна Watch можно при помощи клавиши “Del”, предварительно выделив ее подсветкой.

Используя опцию меню Evaluate/modify или “Ctrl+F4”, можно изменить значение переменной в процессе выполнения отладки, чтобы протестировать алгоритм с новым заданным значением. Окно этой опции “Evaluate and Modify” можно использовать и в качестве калькулятора, если записать выражения с переменными в строке “Expression” и нажать клавишу “Evaluate’ для получения результата в строке Result.

Св. план 2000, поз. 91

Учебное издание

Авторы: Бахирев Андрей Владимирович,
Живицкая Елена Николаевна,
Комличенко Виталий Николаевич,
Соколов Сергей Александрович

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
по курсу "Основы информатики и вычислительной техники"
для студентов специальности Э.01.03.00 в 2 частях.
В 2-х частях
Часть 1.

Редактор Е.Н. Батурчик

Подписано в печать Формат 60x84 1/16.
Бумага Печать офсетная. Усл. печ. л.
Уч.-изд. л. 2,6. Тираж 120 экз. Заказ

Белорусский государственный университет информатики и радиоэлектроники
Отпечатано в БГУИР. Лицензия ЛП №156. 220013, Минск, П. Бровки, 6