**Херсонский национальный технический университет**

# Лабораторная работа №1, 2

**по дисциплине**

**«Программирование»**

Выполнила

студентка группы 1зКСС

Малахова Т.Н.

Проверил

Дроздова Е.А.

**Херсон 2005**

**Лабораторная работа №1**

Управляющие структуры языка «Си»

***Цель работы:***Изучение приемов использования операторов ветвления и операторов циклов в С.

# *Теоретические сведения:*

Управляющие структуры или операторы управления служат для управления последовательностью вычислений в программе. Операторы ветвления и циклы позволяют переходить к выполнению другой части программы или выполнять какую-то часть программы многократно, пока удовлетворяется одно или более условий.

*Блоки и составные операторы.*

Любая последовательность операторов, заключенная в фигурные скобки, является составным оператором (блоком). Составной оператор не должен заканчиваться (;), поскольку ограничителем блока служит сама закрывающаяся скобка. Внутри блока каждый оператор должен ограничиваться (;).

Составной оператор может использоваться везде, где синтаксис языка допускает применение обычного оператора.

*Пустой оператор.*

Пустой оператор представляется символом (;), перед которым нет выражения. Пустой оператор используют там, где синтаксис языка требует присутствия в данном месте программы оператора, однако по логике программы оператор должен отсутствовать.

Необходимость в использовании пустого оператора часто возникает, когда действия, которые могут быть выполнены в теле цикла, целиком помещаются в заголовке цикла.

*Операторы ветвления.*

К операторам ветвления относятся ***if, if else, ?, switch*** и ***goto***. Общий вид операторов ветвления следующий:

***if (логическое выражение)***

***оператор;***

***if (логическое выражение)***

***оператор\_1;***

***else***

***оператор\_2;***

***<логическое выражение> ? <выражение\_1> : <выражение\_2>;***

Если значение логического выражения истинно, то вычисляется выражение\_1, в противном случае вычисляется выражение\_2.

***switch (выражение целого типа)***

***{***

***case значение\_1:***

***последовательность\_операторов\_1;***

***break;***

***case значение\_2:***

***последовательность\_операторов\_2;***

***break;***

***. . .***

***case значение\_n:***

***последовательность\_операторов\_n;***

***break;***

***default:***

***последовательность\_операторов\_n+1;***

***}***

Ветку ***default*** можно не описывать. Она выполняется, если ни одно из вышестоящих выражений не удовлетворено.

*Оператор цикла.*

В Турбо Си имеются следующие конструкции, позволяющие программировать циклы: ***while, do while*** и ***for***. Их структуру можно описать следующим образом:

Цикл с проверкой условия наверху:

***while ( логическое выражение)***

***оператор;***

Цикл с проверкой условия внизу:

***do***

***оператор;***

***while (логическое выражение);***

Универсальный оператор цикла:

***for (инициализация; проверка; новое\_значение)***

***оператор;***

# *Задание:*

Составить и получить распечатку программы выбора всех четных чисел от 1 до 1000.

Составить и получить распечатку программы выбора всех чисел, заканчивающихся на цифру 5, от 1 до 1000.

Написать программу с использованием цикла ***do-while***, которая ожидает нажатия клавиши ‘ъ’ и по нажатии её завершает работу.

***Программа 1:*** распечатывает все четные числа от 1 до 1000:

#include <stdio.h>

int vit;

int main( void )

{

vit = 2;

while (vit <= 1000)

{

vit = vit +2;

printf("%d\n", vit);

}

system("PAUSE");

return 0;

}

***Программа 2:*** распечатывает все числа от 1 до 1000, которые заканчиваются на цифру 5:

#include <iostream.h>

#include <stdlib.h>

int den;

int main()

{

den = 5;

while (den <= 1000)

{

den = den +10;

printf("%d\n", den);

}

system("PAUSE");

return 0;

}

# *Программа 3:*Программа ожидает нажатия клавиши «Ъ» и по её нажатии завершает работу:

#include <iostream.h>

#include <conio.h>

#include <stdio.h>

//#include <bios.h>

int getkey;(void);

{

int key, lo, hi;

key=bioskey(0);

lo = key & 0X00FF;

hi = (key & 0X0FF00) >> 8;

return((lo==0) ? hi +256: lo);

}

main()

{

clrscr();

int input;

do

{

input = getkey();

printf("Program is running \n");

}

while (input!= ']');

}

**Лабораторная работа N2**

**Программирование с использованием указателей**

***Цель работы:*** Oзнакомиться с понятием указателя, научиться использовать их при программировании на С.

***Теоретические сведения***

Указатель - это переменная, которая содержит в памяти адрес данных. Переменная-указатель содержит местоположение значения. То есть, переменная-указатель указывает на значение, так как она содержит его адрес.

Указатели предоставляют эффективные средства доступа и изменения данных. Так как указатели содержат адреса данных, то при поиске значений в памяти у компьютера сокращается объем работы. Указатели не привязывают данные к какому-либо определенному имени переменной. Они могут содержать адрес любого не именованного значения.

Суть переменных-указателей.

Так как указатели являются обычными переменными, для них действительны все правила именования обычных переменных. Как и в случае с обычной переменной, нужно сначала объявить указатель и только потом его использовать. В Си указатели могут быть на все существующие в языке типы данных; можно создать указатель на целое, символьное данное, и так далее. В зависимости от того, в каком месте программы объявлен указатель, он может быть локальным или глобальным (как и для обычных переменных, использовать глобальные без нужды не рекомендуется).

Единственная разница между обычными переменными и указателями заключается в их содержимом. Указатели содержат не само значение, а его адрес.

В Си имеет два оператора, относящихся к указателям:

& - оператор "адрес значения"

\* - оператор "значение по адресу"

Объявление указателей.

Если нужно объявить переменную для хранения, например, возраста то можно сделать это следующим образом:

***int age = 20;***

Такое объявление переменной ***age*** подразумевает несколько моментов. Во-первых, сообщается Си, что нужна переменная с именем ***age***, и Си резервирует для этой переменной место в памяти. Во-вторых, Си узнает, что ***age*** будет использоваться для хранения только целых чисел. В-третьих, при объявлении эта переменная инициализируется значением 20.

Пусть нужно объявить переменную-указатель, которая не содержит возраст, а указывает на ***age***, переменную, где находятся нужные данные. Для объявления указателя на переменную ***age*** нужно сделать следующее:

***int \*p\_age;***

В этой строке резервируется место для переменной с именем ***p\_age***. Однако это не обычная целочисленная переменная. Так как перед ней стоит \*, Си определит, что это переменная-указатель.

Присваивание значений указателям.

Указатель может содержать адреса значений только соответствующего ему типа. Например, ***p\_age*** может указывать только на целочисленные переменные. Си не инициализирует указатели при их объявлении. Если ***age*** объявлена как показано выше, и нужно, чтобы ***p\_age*** содержала адрес ***age***, нужно присвоить его переменной ***p\_age***:

***p\_age=&age;***

Вместо занесения адреса переменной ***age*** в переменную ***p\_age*** при помощи оператора присваивания можно одновременно объявлять и инициализировать указатели.

***int age=20;***

***int \*p\_age=&age;***

Можно присваивать различные значения переменной ***age*** следующим оператором:

***age=35;***

Либо можно сделать то же самое другим путем:

***\*p\_age=35;***

Эта строка подразумевает "взять ячейку памяти, на которую указывает ***p\_age*** и занести туда значение 35".

Объявление массивов указателей.

Если нужно зарезервировать большое количество указателей для различных данных, можно объявить массив указателей, причем каждый его элемент будет являться указателем одного и того же указанного типа. Следующий пример резервирует массив из 10 указателей на переменную целого типа:

***int \*iptr[10];***

Можно присвоить адрес любому элементу из ***iptr*** таким же образом, как и любому другому указателю, не входящему в массив:

***iptr[4]=&age;***

В следующей строке резервируется массив из 20 указателей на переменные символьного типа:

***char \*cpoint[20];***

***Задание:***

Составить программу для определения максимального элемента одномерного массива. Массив первоначально заполняется нулями, затем данные вводятся с клавиатуры. Заменить нулем все элементы, равные максимальному. Массив задать при помощи указателя. Зарезервировать память под массив при помощи функций динамического распределения памяти.

***Программа:*** определяет максимальный элемент одномерного массива:

#include<stdio.h>

#include<conio.h>

#include<alloc.h>

int\* arr;

void zero();

void in();

int max();

void out();

void main()

{

arr=(int\*)malloc(10);

clrscr();

zero();

out();

getch();

in();

int maxEl=max();

int i;

for(i=0;i<10;i++)

if(\*(arr+i)==maxEl) \*(arr+i)=0;

else ;

clrscr();

out();

getch();

}

void zero()

{

int i;

for(i=0;i<10;i++)

\*(arr+i)=0;

}

void in()

{

printf("\nEnter elements of array:");

printf("\n");

int i;

for(i=0;i<10;i++)

scanf("\n%i",arr+i);

}

int max()

{

int mx=\*arr;

int i=1;

for(i;i<10;i++)

if(mx<\*(arr+i)) mx=\*(arr+i);

else ;

return mx;

}

void out()

{

int i;

for(i=0;i<10;i++)

{

printf("%i",\*(arr+i));

printf(" ");

}

system("PAUSE");

return 0;

}

***Результат работы программы:*** Массив первоначально заполняется нулями, затем данные вводятся с клавиатуры. Заменяются нулем все элементы, равные максимальному. Массив задаётся при помощи указателя. Память под массив зарезервирована при помощи функций динамического распределения памяти.