Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
Санкт-Петербургский горный университет

Кафедра системного анализа и управления

ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ

Методические указания к лабораторным работам для студентов бакалавриата направления 27.03.04

САНКТ-ПЕТЕРБУРГ
2023

УДК 004.43 (073)

ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ: методические указания к лабораторным работам / Санкт-Петербургский горный университет. Сост.: *Ю.В. Ильюшин, Т.В. Кухарова*. СПб, 2023. 36 с.

В методических указаниях содержатся краткие теоретические сведения и методические рекомендации к лабораторным работам по дисциплине «Программирование и основы алгоритмизации».

Предназначены для подготовки студентов бакалавриата по направлению подготовки 27.03.04 «Управление в технических системах».

Научный редактор проф. *Д.А. Первухин*

Рецензент проф. *И.М. Першин* (Северо-Кавказский федеральный университет)

ВВЕДЕНИЕ

Курс «Программирование и основы алгоритмизации» является одним из курсов, направленных на формирование общепрофессиональных и профессиональных компетенций при подготовке специалистов по профилю «Информационные технологии в управлении» направления 27.03.04 – «Управление в технических системах». Изучение данного курса является необходимым элементом при подготовке высококвалифицированных кадров.

Основной целью выполнения лабораторных работ по дисциплине «Программирование и основы алгоритмизации» является получение обучающимися навыков разработки программ, включая анализ условия задачи, разработку оптимального алгоритма программы, применение целесообразных средств языка программирования, тестирование и отладку программы. В методических рекомендациях кратко изложены основные теоретические сведения по каждой из тем, приведены примеры и задания для самостоятельного выполнения.

По результатам выполнения лабораторных работ обучающиеся формируют отчеты.

Отчет о выполнении лабораторной работы должен содержать:

– цель работы;

– задания на лабораторную работу;

–блок-схему или описание алгоритма;

– листинг программы на языке С++, реализующей разработанный алгоритм;

– результаты тестирования программы;

– выводы.

ЛАБОРАТОРНАЯ РАБОТА №1. ОСНОВЫ ПРОГРАММИРОВАНИЯ НА С++

Структура программы на С++

Программа на языке С++ представляет собой последовательность операторов, разделенных между собой точкой с запятой. Программа может содержать комментарии. Все символы, стоящие после двойного слэша // до конца строки, являются комментарием.

Каждая программа включает в себя декларативные части, где описываются (объявляются) используемые данные и функции, и блок действий программы. Часто используемые данные и функции описаны в специальных файлах, называемых заголовочными. Заголовочные файлы имеют расширение .h. В простых программах обычно используется заголовочный файл **iostream.h**. Чтобы включить заголовочный файл в программу используется директива:

#include <имя заголовочного файла>,

которая указывается в самом начале программы, например:

#include <iostream.h>

Главной функцией программы является функция **main()**. В ней содержатся декларативные части с описанием данных, используемых внутри функции **main()**, и блок действия программы. Декларативные части могут следовать в произвольном порядке, но любая переменная должна быть описана до того, как она будет использована. Обычно функция **main()** завершается оператором **return(0)**, который возвращает операционной системе нулевое значение в случае успешного завершения программы. Текст (тело) функции **main()** заключается в фигурные скобки.

В разделах описания данных описываются типы используемых переменных.

Типы переменных:

целые (**int**) – эти переменные могут принимать только целые значения, например, 1,10, -8, +40;

вещественные (**double**) – эти переменные могут принимать вещественные (со знаком) значения, например, 10.5, -8.15;

символьные (**char**) – этим переменным могут присваиваться символьные значения, например, ‘A’ – символ A, ‘d’ – символ d.

Оператор присваивания

Оператор присваивания = обычно используетя следующим образом:

<имя переменной> = <арифметическое выражение>;

При этом осуществляется вычисление <арифметического выражения> и присваивание результата <имени переменной>. В <арифметическом выражении> могут использоваться следующие операции:

* – умножение;
/ – деление;
\+ – сложение;
– – вычитание.

Выполнение операций осуществляется слева направо с соблюдением старшинства операций:

1. Умножение, деление;
2. Сложение, вычитание.

Если необходимо изменить порядок выполнения операций, то используются круглые скобки.

Инкремент и декремент

Оператор инкремента ++ увеличивает операнд на 1.

Запись i++ эквивалентна i = i + 1.

Оператор декремента -- уменьшает операнд на 1.

Запись i-- эквивалентна i = i – 1.

Операторы ввода-вывода

Для выполнения операций ввода-вывода информации во время выполнения программы используются следующие операторы:

cin>>X;

считывает значение, введенное с клавиатуры, и присваивает его переменной **X;**

cout<<Y;

выводит на экран дисплея значение переменной **Y**.

Оператор

cout<<”*текст***\n”;**

выводит на экран *текст*. Форматирующий символ **\n** используется для перехода на следующую строку.

Пример.

Составить программу для вычисления суммы двух целых чисел, значения которых вводятся с клавиатуры.

Текст программы:

```
// включение заголовочного файла
#include <iostream.h>

// главная функция программы
main () {
// Описание переменных
int number1, number2, sum;

// Вывод на экран приглашения для
// ввода 1-го числа
cout<<” Enter the first number :”;

// Считывание значения с клавиатуры в
// переменную number1
cin>>number1;

// Вывод на экран приглашения для
// ввода 2-го числа
cout<<” Enter the second number :”;

// Считывание значения с клавиатуры в
// переменную number2
cin>>number2;

// Вычисление суммы
sum=number1+number2;
```

```
// Вывод на экран результата
cout<<”Sum=”<<sum<<”\n”;

// возврат из функции main()
return(0);
}
```

Задания на лабораторную работу

Задание 1. Даны *x, y*. Составить программу вычисления значения выражения:

a) $\frac{|x|-|y|}{1+|xy|}$ b) $\frac{\sqrt{x^2+y^2}}{xy}$ c) $\frac{x-y}{|x|-|y|}$ d) $\frac{\sqrt{|x|+|y|}}{\sqrt{x^2+1}}$

Задание 2. Составить программу для решения следующей задачи:

a) Дана длина ребра куба. Найти объем куба и площадь его боковой поверхности.
b) Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.
c) Вычислить высоту треугольника, опущенную на сторону *a*, по известным значениям длин его сторон *a, b, c*.
d) По данным сторонам прямоугольника вычислить его периметр, площадь и длину диагонали.

Задание 3. Ученик начал решать задачи данного урока программирования, когда электронные часы показывали h_1 часов и min_1 минут, а закончил, когда было h_2 часов и min_2 минут. Составьте программу, позволяющую определить, сколько времени (в часах и минутах) ученик решал эти задачи.

ЛАБОРАТОРНАЯ РАБОТА № 2. РАЗВЕТВЛЯЮЩИЕСЯ ПРОЦЕССЫ

Условный оператор

Условный оператор может иметь одну из следующих форм записи:

if (<выражение>**)**
    <оператор1>**;**
else
    <оператор2>**;**

если <выражение> истинно (его значение ненулевое), то выполняется <оператор1>, в противном случае – <оператор2>;

if (<выражение>**) {**
    <оператор 1>**;**
    <оператор 2>**;**
    …
    <оператор N>**;**
    }
else {
    <оператор 1'>**;**
    <оператор 2'>**;**
    …
    <оператор N'>**;**
    }

если <выражение> истинно, то выполняются <оператор 1>, <оператор 2>, …, <оператор N>, в противном случае – <оператор 1'>, <оператор 2'>, …, <оператор N'>.

Вторая часть условного оператора – от слова **else** – может отсутствовать. В этом случае, если <выражение> ложно, то выполняется оператор, следующий в программе за условным оператором.

Пример.

```
if (x <= 1)
        z = x+1;
else
        z = x-1;
```

Оператор ветвления

Оператор ветвления имеет вид:

switch (<выражение>**) {**
case <значение 1>**:**
    <оператор1>**;**
    break;
case <значение 2>**:**
    <оператор2>**;**
    break;
…
case <значение N>**:**
    <операторN>**;**
    break;
default :
    <оператор по умолчанию>**;**
}

Если <выражение> равно <значению 1>, то выполняется <оператор1>, если <выражение> равно <значению 2>, то выполняется <оператор2>, …, если <выражение> равно <значению N>, то выполняется <операторN>; если <выражение> не равно ни одному из перечисленных значений, то выполняется <оператор по умолчанию>.

Пример.

```
switch (Day) {
case 1 :
        cout << “It is Monday today” \n;
        break;
case 2 :
```

```
        cout << “It is Tuesday today \n”;
        break;
case 3 :
        cout << “It is Wednesday today \n”;
        break;
case 4 :
        cout << “It is Thursday today \n”;
        break;
case 5 :
        cout << “It is Friday today \n”;
        break;
case 6 :
        cout << “It is Saturday today \n”;
        break;
case 7 :
        cout << “It is Sunday today \n”;
        break;
default :
        cout << “Illegal day \n”;
}
```

Задания на лабораторную работу

Задание 1. Используя оператор *if*, вычислить заданное выражение для данных типа *Integer:*

a)
$$f(x)=\begin{cases} x^2+3 & ,x<0 \\ 6\sqrt{x} & ,0\le x\le 5 \\ -x+9 & ,x>5 \end{cases}$$

b)
$$f(x)=\begin{cases} 3|x| & ,x<-2 \\ 9x & ,-2\le x\le 2 \\ \sin x & ,x>2 \end{cases}$$

c)
$$f(x)=\begin{cases}\sin^2 x-\cos^2 x & ,x<0\\ \ln(3x+2) & ,0\le x\le 2\\ x^2-x^3 & ,x>2\end{cases}$$

d)
$$f(x)=\begin{cases}|2x-2| & ,x<-2\\ \sin x & ,-2\le x\le 5\\ x^4 & ,x>5\end{cases}$$

Задание 2. Найти алгоритм решения задачи и реализовать его с помощью оператора (операторов) if-then-else:

a) Составить программу, реализующую эпизод сказки: машина спрашивает, куда пойдет герой, и в зависимости от ответа (налево – (-1), прямо – 0, направо – 1), печатает, что произойдет с героем.

b) В Атлантическом океане терпит бедствие пассажирский теплоход «Посудина». Все пассажиры будут спасены, если на помощь успеют два судна. Судно продержится на плаву *t* часов. Скорость судов-спасателей 40 узлов. Составить программу, определяющую спасутся ли пассажиры. Известны расстояния от трех судов-спасателей до тонущего судна.

c) Через старый мост движется поток автомашин. Одновременно на мосту могут находиться 3 машины. Если на мост въедут 3 легковых или 2 легковых и грузовик – мост выдержит. Если 2 грузовика и легковая или 3 грузовика – рухнет.

Задание 3. Используя оператор выбора, составить программы решения следующих задач.

a) По номеру дня недели вывести на печать рабочий это день или выходной, считая выходными субботу и воскресенье.

b) По номеру месяца указать, к какому времени года он относится.

c) По номеру месяца вывести на печать количество дней в нем.

d) Единицы массы пронумерованы следующим образом: 1 — килограмм, 2 — миллиграмм, 3 — грамм, 4 — тонна. Дан номер

единицы массы и масса тела *M* в этих единицах (*M* - вещественное число). Вывести массу данного тела в килограммах.

Задание 4. Даны действительные числа *a, b, c, x, y*. Выяснить, пройдет ли кирпич с ребрами *a,b,c* в прямоугольное отверстие со сторонами *x* и *y*. Просовывать кирпич в отверстие разрешается только так, чтобы каждое из его ребер было параллельно или перпендикулярно каждой из сторон отверстия.

Задание 5. Составить программу для решения следующей задачи: сможет ли шар радиуса *R* пройти в ромбообразное отверстие со стороной *P* и острым углом *Q*?

ЛАБОРАТОРНАЯ РАБОТА № 3. ЦИКЛИЧЕСКИЕ ПРОЦЕССЫ

Оператор цикла while

Синтаксис оператора цикла **while**:

while (<выражение>**)**
{<оператор1>**;**
<оператор2>**;**
…
<операторN>**;**
}

Пока <выражение> истинно, выполняются <оператор 1>, <оператор 2>, …, <оператор N>.

Следует отметить, что если <выражение> всегда истинно, то цикл будет бесконечным. Если <выражение> с самого начала ложно, то ни один из операторов не будет выполнен.

Пример.

Программа выводит на экран целые числа от 1 до 10.

```
#include <iostream.h>
main() {
int number;
```

```
number = 1;
while (number <= 10) {
cout << “number = ” << number<< “\n”;
number++;
}
return (0);
}
```

Оператор цикла do while

Синтаксис оператора цикла **do while**:

do {
<оператор1>**;**
<оператор2>**;**
…
<операторN>**;**
}
while (<выражение>**);**

Сначала выполняются <оператор1>, <оператор2>, …, <операторN>, затем проверяется <выражение>. Операторы будут выполняться до тех пор, пока <выражение> будет истинным.

Следует отметить, что операторы всегда выполняются по крайней мере один раз, так как проверка <выражения> осуществляется после выполнения операторов.

Пример.

Программа выводит на экран целые числа от 1 до 10.

```
#include <iostream.h>
main() {
int number=0;
do {
number++;
cout << “number = “ << number<< “\n”;
}
```

```
while (number <10);
return (0);
}
```

Оператор цикла for

Оператор цикла **for** имеет вид:

for (<выражение 1>**;** <выражение 2>**;** <выражение 3>**) {**
<оператор1>**;**
<оператор2>**;**
…
<операторN>**;**
}

Этот оператор цикла эквивалентен следующей записи:
<выражение 1>**;**
while
(<выражение 2>**) {**
<оператор1>**;**
<оператор2>**;**
…
<операторN>**;**
}

Пример.

Вывод на экран целых чисел от 1 до 10.

```
for (number=1; number <=10; number++)
cout << “number = “<<number<<”/n”;
```

Задания на лабораторную работу

Задание 1. Целочисленная арифметика.

Найти количество натуральных двузначных чисел, каждое из которых делится на 3 и на 13.

а) Найти количество натуральных четырехзначных чисел, каждое из которых не делится ни на 2, ни на 3.

b) Найти количество натуральных чисел, не превосходящих 1000, каждое из которых кратно 25 и не кратно 3.

c) Найти те натуральные числа, не превосходящие *x,* которые при делении на 10 дают в остатке 5.

d) Найти те натуральные числа, не превосходящие *x,* которые при делении на 3 дают в остатке 2.

Задание 2. Найти алгоритм решения задачи и реализовать его в виде программы.

a) Начальный вклад в банк составляет *a* рублей. Через сколько лет он станет больше *b* рублей? Каждый год вклад увеличивается на 3%.

b) Ежегодный прирост рыбы в пруду составляет 15%. Запасы рыбы оценены в *A* тонн. Ежегодный план отлова *B* тонн. Подсчитать, сколько лет можно выдерживать заданный план?

c) Каждая бактерия делится на две в течение одной минуты. В начальный момент имеется *A* бактерий. Сколько времени потребуется, чтобы количество бактерий превзошло *X*?

d) Определить количество посетителей салона, которых успеет обслужить мастер-стилист, если его рабочий день составляет *t* часов и известна продолжительность (в минутах) обслуживания каждого посетителя очереди (вводится пользователем).

Задание 3. Составить программу для решения следующей задачи: вычислить количество точек с целочисленными координатами, попадающими в круг радиуса R (R>0) с центром в начале координат.

ЛАБОРАТОРНАЯ РАБОТА № 4. МАССИВЫ

Массив – это упорядоченный набор элементов одного и того же типа. Каждый массив должен иметь имя. При объявлении массива перед его именем указывается тип его элементов, а после его имени – количество элементов массива в квадратных скобках. Например, в строке

```
int mas [100];
```

объявляется массив целых чисел, состоящий из 100 элементов.

Нумерация элементов массива начинается с 0. Номер элемента массива называется *индексом*. Таким образом, первый элемент массива имеет индекс 0. Массив, объявленный как name[N] имеет допустимое значение индексов от 0 до N-1; name[23] обозначает 24–ый элемент массива. Элементы массива запоминаются в памяти последовательно.

Массивы могут обладать двумя и более измерениями. Например, в строке

```
int S[25][80];
```

объявляется массив из 25 строк, состоящих из 80 элементов;
S[4] – 5-ая строка;
S[4][2] – 3-ий элемент 5-ой строки.
Двумерные массивы заполняются в памяти построчно.

Строки

Строка – это символьный массив. Последним элементом строки должен быть нулевой символ, например:

```
char str[4];
str[0]='c';
str[1]='a';
str[2]='r';
str[3]='\0';
```

или

```
char str[4]="car";
```

В последнем случае ноль в конце строки подставляется автоматически.

Пример.

Программа осуществляет сложение двух матриц размером 3x4, одна из которых постоянная, вторая вводится пользователем с клавиатуры.

```
#include <iostream.h>
#define M  4
#define N  3
//Вычисляем  c = a + b, где a – постоянная матрица,
//матрица b  вводится с клавиатуры

main() {
int i, j, b[N][M], c[N][M];

//  Задание элементов постоянной матрицы a
int a[N][M]={1,3,4,6,4,-1,0,5,7,-4,12,7};

// Ввод матрицы b
for (i=0; i<=N-1; i++)
for (j=0; j<=M-1; j++) {
cout<< "enter b ["<<i<<"]["<<j<<"]:";
cin>>b[i][j];
}

// Сложение матриц
for (i=0; i<=N-1; i++)
for (j=0; j<=M-1; j++)
c[i][j]=a[i][j]+b[i][j];

//Вывод матрицы c на экран
for (i=0; i<=N-1; i++) {
for (j=0; j<=M-1; j++)
cout << c[i][j];
cout <<"\n";
}
```

```
return (0);
}
```

В программе присутствует директива define, позволяющая сопоставить некоторому идентификатору определенное значение, которое будет использоваться в продолжение работы всей программы. При компиляции программы этот идентификатор автоматически заменяется на его значение.

Задания на лабораторную работу

Задание 1. Создать программу, вычисляющую сумму элементов одномерного массива целых чисел из 10 элементов. Массив вводится с клавиатуры.

Задание 2. Создать программу для решения следующей задачи:

а) Определить в каком из трех массивов больше среднее арифметическое элементов, меньших заданного числа. Если в двух или трёх массивах значения среднего арифметического совпадают, вывести соответствующее сообщение.

б) Даны две матрицы разного размера. Для той из матриц, в которой меньше среднее арифметическое положительных элементов, найти произведение ненулевых элементов в каждой строке.

Задание 3. Создать программу, вычисляющую произведение постоянной матрицы размерности 3×3 и вектора целых чисел из 3-х элементов, которые вводятся с клавиатуры.

Задание 4. Создать программу, вычисляющую произведение двух матриц, размеры которых задаются директивами препроцессора.

Задание 5. Создать программу, осуществляющую сортировку элементов одномерного массива из 10 элементов по убыванию.

Задание 6. Создать программу, осуществляющую ранжирование элементов одномерного массива из 10 элементов. Ранжирование – процедура присвоения каждому элементу ряда его

номера в ряду, упорядоченному по возрастанию. При наличии одинаковых элементов их ранг определяется как среднее арифметическое их номеров в упорядоченном ряду.

ЛАБОРАТОРНАЯ РАБОТА № 5. ФУНКЦИИ

Использование функций

Отдельные части программы можно оформить в виде функций. Функции позволяют разделить большую программу на составляющие. Каждая функция выполняет свою задачу. Функции часто используются для выполнения повторяющихся операций.

Описание функции имеет вид:

<тип возвращаемого значения> <имя функции>
(<параметры>)
{
<тело функции>
}

Первая строка называется *заголовком функции*. После нее не ставится символ **;** (точка с запятой). Параметры в заголовке функции являются формальными и перечисляются через запятую с указанием их типов. Функция может возвращать в программу некоторое значение после завершения своей работы (в этом случае ее работа завершается оператором возврата **return()**), а может ничего не возвращать. В первом случае в качестве <типа возвращаемого значения> указывается конкретный тип, во втором случае используется ключевое слово **void**. Если функция не имеет параметров, то в круглых скобках указывается ключевое слово **void**. <тело функции> содержит декларативные части, описывающие используемые в функции переменные, и блок действия функции.

Описания всех функций указываются в конце программы, после завершения текста главной функции **main()**.

Заголовки всех функций должны быть перечислены перед функцией **main()**. Они носят названия *прототипов* функций. Каждый прототип заканчивается символом **;** (точка с запятой).

Вызов функции представляет собой имя функции с указанием в круглых скобках фактических параметров (без типов), перечисленных в порядке, соответствующем перечислению формальных параметров в заголовке и прототипе функции.

Пример.
Сложение двух целых чисел с использованием функции.

```
#include <iostream.h>

// Прототип функции
int Summing(int x, int y);

main() {
int number1, number2, sum;
// Ввод первого числа
cout << “Enter the first number: “;
cin >> number1;

// Ввод второго числа
cout << “Enter the second number: “;
cin >> number2;

// Вычисление суммы
sum = Summing(number1, number2);

// Вывод результата на экран
cout << “sum=”<<sum<<”\n”;

return (0);
}

// Описание функции Summing
int Summing (int x, int y) {
int S;
S=x+y;
```

```
return(S);
}
```

Рекурсивные функции

Рекурсивная функция – это функция, вызывающая саму себя.

Пример.

Функция вычисления факториала.

```
int Factorial(int n) {
if (n>1)
return(n*Factorial(n-1));
else return (1);
}
```

Здесь используется формула вычисления факториала:
n! = (n-1)!*n.

Встраиваемые функции

Встраиваемая функция объявляется с ключевым словом **inline**. Вместо прототипа встраиваемой функции указывается ее полное описание. Обычно встраиваемыми объявляются функции с небольшим кодом. При компиляции тело встраиваемой функции автоматически подставляется в месте вызова этой функции.

Пример.

Описание встраиваемой функции, которая возвращает большее из двух значений.

```
inline int max(int a, int b) {
if (a>=b) return (a);
else return (b);
}
```

Перегруженные функции

В C++ функции могут иметь одинаковые имена, если они отличаются хотя бы одним параметром. Такие функции называются *перегруженными*. Перегруженные функции часто используются, когда необходимо произвести аналогичные действия над объектами разных типов.

***Пример*.**

Вычисляются квадраты целого и вещественного чисел. Здесь функция **square()** является перегруженной.

```
#include <iostream.h>
//Прототипы функций
int square(int a);
double square(double a);

main() {
int x; double y;
// Ввод чисел
cout << “Enter an integer number: “;
cin >> x;
cout << “Enter an real number: “;
cin >> y;

//Вывод результатов
cout <<”x*x = ” << square(x) << “\n”;
cout <<”y*y = ” << square(y) << “\n”;

return (0);
}

// Описание функций
int square(int a) {
return (a*a);
}
```

```
double square(double a) {
return (a*a);
}
```

Задания на лабораторную работу

Задание 1. Вычислить сумму элементов двумерного массива размером 2x3. Элементы массива вводятся с клавиатуры. суммирование должно осуществляться в функции.

Задание 2. Сосчитать определенный интеграл по формуле трапеций:

$$\int_a^b f(x)dx = \frac{h}{2}(y_0 + 2y_1 + 2y_2 + ... + 2y_{n+1} + y_n),$$

где $h = \frac{b-a}{n}$,

$y = f(x_i)$,

$x_i = a + h \cdot i$,

$i = 0, 1, \ldots, n$.

Числа a, b, n вводятся с клавиатуры.

Вычисление значений $f(x_i)$ оформить как функцию.

Расчет выполнить для $f(x) = 2x^2 + x + 3$.

Задание 3. Реализовать вычисление реккурентной формулы

$$y(t) = \frac{y(t-\Delta t)\cdot T + K\cdot X\cdot \Delta t}{T+\Delta t}$$

с помощью рекурсивной функции.

Задание 4. Реализовать с помощью перегруженных функций вычисление площади поверхности призмы с квадратным основанием для действительных и целых параметров.

ЛАБОРАТОРНАЯ РАБОТА № 6. СТРУКТУРЫ

Структура – это комбинированный тип данных, состоящий из нескольких компонентов, которые могут быть разного типа (в отличие от массива). Эти компоненты называются членами

структуры. Структуры могут включать в себя переменные любых типов, массивы и другие структуры.

Чтобы объявить структуру, надо перед именем структуры указать ключевое слово **struct**, а после имени в фигурных скобках перечислить члены структуры с указанием их типов:

struct < имя структуры> **{**
<тип члена> <имя члена>**;**
…
<тип члена> <имя члена>**;**
};

После закрывающей фигурной скобки необходим символ **;** (точка с запятой).

Пример структуры, содержащей координаты точки на плоскости:

```
struct coordinate {
double x;
double y;
};
```

Далее возможно объявление структурных переменных типа **coordinate**:

```
coordinate point1, point2;
```

Для доступа к членам структуры используется конструкция вида:

<имя структурной переменной>**.**<имя члена>

Например:

```
point1.x = 5.2;
point2.y = point1.y+3.5;
```

Структуры могут быть вложенными, то есть членом структуры может быть другая структура. Например,

```
struct address {
int hiuse;
char street [50];
char city [20];
char state [20];
char zipcode [10];
};

struct person {
char lastName[20];
char firstName[10];
address homeAddress;
address workAddress;
};
```

Объявление структурной переменной типа **person**:

```
person thePerson;
```

Доступ к членам структурной переменной **thePerson**:

```
thePerson.homeaddress.house = 15;
```

Можно объявить массив структур типа **person**:

```
person thePerson [30];
```

В этом случае доступ к членам структур будет осуществляться следующим образом:

```
thePerson[4].homeaddress.house = 23;
```

Пример.

Программа позволяет ввести, отредактировать и вывести на экран данные сотрудников фирмы.

```
#include <iostream.h>
```

```
// MAX – максимальное число сотрудников фирмы
#define MAX  25

// Структура address содержит адрес:
// название улицы, номер дома, номер квартиры
struct address {
char street [20];
int house;
int flat;
};

// Структура person содержит имя, фамилию
// и домашний адрес сотрудника
struct person {
char firstName [10];
char lastName [20];
address homeAddress;
};

// Прототипы функций
person Init(void);
void toScreen (person Pers);
address changeAddr (void);

main () {
int n, i;
// Объявляется массив структур
person thePerson[MAX];

// Ввод данных о сотрудниках
for (i=0; i<MAX; i++)
thePerson[i] = Init();

// Вывод на экран данных о сотрудниках
for (i=0; i<MAX; i++)  toScreen(thePerson[i];
```

```
// Изменение адреса сотрудника
// с заданным порядковым номером
cout<<”Enter a number:”;
cin>>n;
thePerson[n-1].homeAddress = changeAddr();

// Вывод на экран измененных данных
// о сотрудниках
for (i=0; i<MAX; i++)
toScreen (thePerson[i]);
return(0);
}

// Описания функций

// Функция Init() позволяет ввести с клавиатуры
// данные одного сотрудника.
person Init(void) {
person newPers;
cout<<”Enter the first name:”;
cin>>newPers.firstName;
cout<<”Enter the last name:”;
cin>>newPers.lastName;
cout<<”Enter the street:”;
cin>>newPers.homeAddress.street;
cout<<”Enter the house number:”;
cin>>newPers. homeAddress.house;
cout<<”Enter the flat number:”;
cin>>newPers.homeAddress.flat;
return(newPers);
}

// Функция toScreen() выводит на экран
// данные одного сотрудника
void toScreen(person Pers) {
cout << Pers.firstName<<”  “
```

```
<< Pers.lastName<<” “
<< Pers.homeAddress.street<<” “
<< Pers.homeAddress.house<<” “
<< Pers.homeAddress.flat<<”\n “
}

// Функция changeAddr() позволяет ввести
// с клавиатуры новый адрес сотрудника
address changeAddr(void) {
address newAddress;
cout<<”Enter the new street:”;
cin >> newAddress.street;
cout<<”Enter the new house number:”;
cin >> newAddress.house;
cout<<”Enter the new flat number:”;
cin >> newAddress.flat;
return(newAddress);
}
```

Задание на лабораторную работу

Написать программу для ввода, редактирования (изменения даты последней продажи) и вывода на экран данных об автомобилях, выставленных на продажу в комиссионном автомагазине, с указанием марки автомобиля, даты выпуска (день, месяц, год), даты последней продажи выпуска (день, месяц, год), стоимости автомобиля. Вычислить общую стоимость всех автомобилей. Использовать вложенные структуры.

ЛАБОРАТОРНАЯ РАБОТА № 7. УКАЗАТЕЛИ

Распределение памяти

В C++ имеются два способа распределения памяти под переменные: статический и динамический. При статическом распределении всем объявленным переменным выделяются фиксированные участки памяти, которые закрепляются за

переменными на все время работы программы. Такие переменные называются *статическими.*

При динамическом распределении памяти участки памяти под переменные могут выделяться, а затем освобождаться во время выполнения программы. такие переменные называются *динамическими*. Динамические переменные размещаются в памяти с помощью указателей и специальных операторов, которые выделяют и освобождают память.

Каждая ячейка памяти имеет *адрес* (номер).

Указатель – это обычная статическая переменная, в которой хранится адрес динамической переменной (говорят, что указатель адресует динамическую переменную). При объявлении перед идентификатором указателя ставится символ *, например:

```
int *p;  // Объявляется переменная p как указатель на целое
```

В отличие от статических переменных при объявлении указателя память для соответствующей динамической переменной сразу не выделяется. Для выделения памяти под динамическую переменную используется оператор **new**:

new (<тип динамической переменной> | <размер блока памяти>);

Оператор **new** возвращает адрес начала блока выделенной памяти. Например, в строке

p = new int;

выделяется память под целое число, и адрес выделенного блока памяти присваивается указателю **p**.

Для получения доступа к значению, адресуемому указателем, используется *разыменование указателя*, при этом перед идентификатором указателя ставится символ *. Например:

*p = 5;

в данном случае в область памяти, адресуемую указателем p, записывается целое число 5.

Следует отличать указатель и значение, которое он адресует. Например, команда:

cout<<p;

выводит на экран дисплея значение указателя p, то есть адрес ячейки памяти;

cout<<*p;

выводит на экран содержимое ячейки памяти с адресом p, то есть целое число 5.

Для освобождения занятой памяти от динамических переменных используется оператор **delete**:

delete(<указатель на динамическую переменную>);

Например:

delete p;

освобождает область памяти, адресуемую указателем **p**. Параметром оператора **delete** может являться список указателей, перечисленных через запятую. Например,

delete p1,p2,p3;

Указатели и структуры

Чтобы рациональнее использовать память, под структуры желательно выделять место динамически.

Рассмотрим структуру **coordinate**:

```
struct coordinate {
int x;
int y;
};
coordinate *p;   // p – указатель на структуру типа coordinate
```

Для доступа к членам структуры через указатель используется символ ->, например,

```
p->x = 123;
p->y = 321;
```

Пример.

Программа позволяет ввести в память компьютера и затем вывести на экран дисплея данные о сотрудниках фирмы (фамилию, год рождения).

```
#include <iostream.h>
#define MAX 25

struct sotr {
char lastName[20];
int year;
};

main() {
// Объявление массива указателей
// на структуру типа sotr
sotr *ps[MAX];

for (int i=0; i<MAX; i++) {
// Выделение места в памяти под сотрудника
ps[i] = new sotr;
// Ввод данных сотрудника
cout<<”Enter the last name:”;
cin>>ps[i]->lastName;
cout<<”Enter the year:”;
cin>>ps[i]->year;
}

// Вывод данных о сотрудниках на экран
for (int i=0; i<MAX; i++) {
cout<< ps[i]->lastName<<”  “
<< ps[i]->year<<”\n“;

// Освобождение памяти, выделенной под сотрудника
delete ps[i];
}
```

```
return(0);
}
```

Указатели на указатели

Указатель может ссылаться на другой указатель, который содержит адрес обычной переменной. Такой указатель называется указатель на указатель (pointer to pointer), а адресация — косвенной. В случае косвенной адресации один указатель ссылается на другой указатель, где хранится адрес переменной, в котором записано нужное значение.

Чтобы объявить переменную указатель на указатель, перед идентификатором записывается дополнительная звездочка. Чтобы извлечь значение переменной при косвенной адресации, необходимо дважды применить операцию разыменовывания.

***Пример*.**

Программа позволяет создать динамический двумерный массив.

```
#include <iostream.h>
int main() {
double** p; // указатель на указатель

// ввод количества строк m и столбцов n
int m, n ;
cout << "Enter quantity of rows\t-> " ; cin >> m ;
cout << "Enter quantity of columns\t-> " ; cin >> n;

// выделение памяти для массива указателей на строки
p = new double* [m];
// выделение памяти для элементов строк
for ( int i = 0; i < m; i++ ) p[i] = new double [n];

// заполнение матрицы значениями
for ( int i = 0; i < m; i++ )
for ( int j = 0; j < n; j++ ) cin >> p[i] [j];
```

```
// вывод матрицы
for ( int i = 0; i < m; i++ ) {
for ( int j = 0; j < n; j++ )
cout << p[i] [j] << '\t';
cout << endl;
}

// освобождение памяти
for ( int i = 0; i < m; i++) delete [] p [i];
delete [] p;
return 0;
}
```

Задания на лабораторную работу

Задание 1. Написать программу, вычисляющую сумму двух целых чисел, значения которых вводятся с клавиатуры. В качестве переменных использовать только указатели.

Задание 2. Написать программу для ввода в память компьютера и последующего вывода на экран данных об автомобилях с указанием марки, года выпуска и стоимости. Вычислить общую стоимость всех автомобилей. Использовать массив указателей на структуру.

Задание 3. Написать программу для вычисления произведения двух матриц. Размерности матриц водятся с клавиатуры. Использовать массивы указателей.

РЕКОМЕНДАТЕЛЬНЫЙ БИБЛИОГРАФИЧЕСКИЙ СПИСОК

а) основная литература

1. Основы алгоритмизации и программирования: Учебное пособие / Колдаев В.Д; Под ред. проф.Л.Г. Гагариной - М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2016. - 416 с. [Электронный ресурс] - http://znanium.com/bookread2.php?book=537513

2. *Воронцова Е.А.* Программирование на С++ с погружением: практические задания и примеры кода - М.:НИЦ ИНФРА-М, 2016. - 80 с. [Электронный ресурс] - http://znanium.com/bookread2.php?book=563294

3. *Царев, Р.Ю.* Программирование на языке Си: учеб. пособие / Р.Ю. Царев. – Красноярск: Сиб. федер. ун-т, 2014. – 108 с. [Электронный ресурс] - http://znanium.com/bookread2.php?book=510946

б) дополнительная литература

4. Объектно Ориентированное Программирование. Хорошая книга для Хороших Людей / Комлев Н.Ю. - М.:СОЛОН-Пр., 2015. - 298 с. [Электронный ресурс] - http://znanium.com/bookread2.php?book=884394

5. Программирование на языке Си/А.В.Кузин, Е.В.Чумакова - М.: Форум, НИЦ ИНФРА-М, 2015. - 144 с. [Электронный ресурс] - http://znanium.com/bookread2.php?book=505194

6. Программирование графики на С++. Теория и примеры : учеб. пособие / В.И. Корнеев, Л.Г. Гагарина, М.В. Корнеева. — М. : ИД «ФОРУМ» : ИНФРА-М, 2017. - 517 с. [Электронный ресурс] - http://znanium.com/bookread2.php?book=562914

7. Полезное программирование: Практическое пособие / Комлев Н.Ю. - М.:СОЛОН-Пр., 2016. - 256 с. [Электронный ресурс] - http://znanium.com/bookread2.php?book=902533

8. *Ступина, А.А.* Технология надежностного программирования задач автоматизации управления в технических системах: монография / А. А. Ступина, С. Н. Ежеманская. - Красноярск : Сиб. федер. ун-т, 2011. - 164 с. [Электронный ресурс] - http://znanium.com/bookread2.php?book=442655

9. C++ как второй язык в обучении приемам и технологиям программирования: учебное пособие / Я.М. Русанова. - Ростов-на-Дону: Издательство ЮФУ, 2010. - 200 с. [Электронный ресурс] - http://znanium.com/bookread2.php?book=550811

Содержание

ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ

Методические указания к лабораторным работам для студентов бакалавриата направления 27.03.04

Сост.: *Ю.В. Ильюшин, Т.В. Кухарова*

Печатается с оригинал-макета, подготовленного кафедрой системного анализа и управления

Ответственный за выпуск *Ю.В. Ильюшин*

Лицензия ИД № 06517 от 09.01.2002

Подписано к печати 02.02.2023. Формат 60×84/16.
Усл. печ. л. 2,1. Усл.кр.-отт. 2,1. Уч.-изд.л. 1,9. Тираж 50 экз. Заказ 46.

Санкт-Петербургский горный университет
РИЦ Санкт-Петербургского горного университета
Адрес университета и РИЦ: 199106 Санкт-Петербург, 21-я линия, 2