ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)

Кафедра вычислительных машин, комплексов, систем и сетей

Л.А. Надейкина

ПРОГРАММИРОВАНИЕ

Учебно-методическое пособие
по выполнению практических работ

Часть 2

для студентов I курса
направления 09.03.01
очной формы обучения

Москва
ИД Академии Жуковского
2018

УДК 004.42(07)
ББК 6Ф7.3
Н17

Рецензент:
Черкасова Н.И. – канд. физ.-мат. наук, доц. каф. ВМКСС

Надейкина Л.А.
Н17 Программирование [Текст] : учебно-методическое пособие по выполнению практических работ, ч. 2 / Л.А. Надейкина. – М.: ИД Академии Жуковского, 2018. – 36 с.

Данное учебно-методическое пособие издается в соответствии с рабочей программой учебной дисциплины «Программирование» по учебному плану для студентов I курса направления 09.03.01 очной формы обучения.

Рассмотрено и одобрено на заседании кафедры 08.05.2018 г. и методического совета 16.05.2018 г.

УДК 004.42(07)
ББК 6Ф7.3

В авторской редакции

Подписано в печать 21.06.2018 г.
Формат 60x84/16 Печ. л. 2,25 Усл. печ. л. 2,09
Заказ № 327/0604-УМП09 Тираж 30 экз.

Московский государственный технический университет ГА
125993, Москва, Кронштадтский бульвар, д. 20

Издательский дом Академии имени Н. Е. Жуковского
125167, Москва, 8-го Марта 4-я ул., д. 6А
Тел.: (495) 973-45-68
E-mail: zakaz@itsbook.ru

© Московский государственный технический
университет гражданской авиации, 2018

СОДЕРЖАНИЕ

Содержание занятий. 4
1.Определение и вызов функций. Передача параметров (занятия 6,7) .
1.1. Цель занятия. 4
1.2. Методические указания по теме 4
1.3. Задания для самостоятельного решения 9
1.4. Контрольные вопросы 10
2.Указатель на функцию – параметр функции (занятие 8). 10
2.1. Цель занятия 10
2.2. Методические указания по теме 10
2.3. Задания для самостоятельного решения 14
2.4.Контрольные вопросы 15
3. Реализация рекурсивных алгоритмов (занятие 9). 15
3.1. Цель занятия 15
3.2. Методические указания по теме 15
3.3. Задания для самостоятельного решения 16
3.4. Контрольные вопросы 17
4. Разработка алгоритмов и программ с использованием структур и массивов структур. Динамическое выделение памяти (занятие 10) . . . 17
4.1. Цель занятия 17
4.2. Методические указания по теме 18
4.3. Задания для самостоятельного решения 22
4.4. Контрольные вопросы 23
5. Создание связанных структур - списков для решения различных практических задач (занятие 11). 23
5.1. Цель занятия 23
5.2. Методические указания по теме 23
5.3. Задания для самостоятельного решения 28
5.4. Контрольные вопросы. 28
6. Разработка связанных структур - очередей и функций работы с ними (занятие 12) 28
6.1. Цель занятия 28
6.2. Методические указания по теме 28
6.3. Задания для самостоятельного решения 30
6.4. Контрольные вопросы. 30
7. Алгоритмы сортировки массивов (занятие 13) 30
7.1. Цель занятия 30
7.2. Методические указания по теме 30
7.3. Задания для самостоятельного решения 35
7.4. Контрольные вопросы. 36
Литература 36

СОДЕРЖАНИЕ ЗАНЯТИЙ

1. ОПРЕДЕЛЕНИЕ И ВЫЗОВ ФУНКЦИЙ. ПЕРЕДАЧА ПАРАМЕТРОВ (занятия 6,7)

1.1. Цель занятия

Формирование компетенции (ОПК-2.1.1), в виде получения навыков программирования функций различного назначения, используя алгоритмы структурного программирования. Формирование компетенции (ОПК-5.3.4), в виде получения навыков работы с библиотеками функций и классов. Формирование компетенции (ПК 1.1.1) в виде опыта разработки алгоритмов и программ, с использованием методов отладки. Формирование компетенции (ПК-2.2.3) в виде опыта работы с многомерными статическими и динамическими массивами;

1.2. Методические указания по теме

Программа на языке C++ представляет собой совокупность произвольного количества функций, одна (и единственная) из которых - главная функция с именем ***main***. Выполнение программы начинается и заканчивается выполнением функции ***main*.** Выполнение неглавных функций инициируется в главной функции непосредственно или в других функциях, которые сами инициируются в главной.

Функции – это относительно самостоятельные фрагменты программы, оформленные особым образом и снабженные именем.

Каждая функция существует в программе в единственном экземпляре, в то время как обращаться к ней можно многократно из разных точек программы.

***Определение функции** – это программный текст функции.* Определение состоит из заголовка и тела функции:

<тип > <имя функции > (<список формальных параметров>)

тело функции

1) ***тип*** **–** тип возвращаемого функцией значения, с помощью оператора ***return*,** если функция не возвращает значения, следует поместить слово ***void***;

2) ***имя функции*** **–** идентификатор, уникальный в программе;

3) ***список формальных параметров (сигнатура параметров*)** **–** заключенный в круглые скобки список спецификаций отдельных формальных параметров перечисляемых через запятую:

<***тип параметра> <имя параметра>,***

<тип параметра> <имя параметра> = <умалчиваемое значение>,

если параметры отсутствуют, в заголовке после имени функции должны стоять пустые скобки *( )***;** для формального параметра может быть задано умалчиваемое значение – начальное значение параметра;

4) ***тело функции*** **–** это блок или составной оператор, т.е. последовательность определений, описаний и операторов, заключенная в фигурные скобки.

***Вызов функции* передает ей управление, а также фактические параметры при наличии в определении функции формальных параметров.**

Форма вызова функции:

имя функции (список фактических параметров);

Фактические параметры должны соответствовать формальным параметрам по количеству, типу, и по расположению параметров. При вызове функции происходит передача фактических параметров в функцию, и именно эти параметры обрабатываются функцией. После завершения выполнения всех операторов функция возвращает управление программой в точку вызова.

Оператор return

Оператор **return** *- оператор возврата управления программой и значения в точку вызова функции.* С помощью этого оператора функция может вернуть одно скалярное значение любого типа. Форма оператора:

return (выражение);

- выражение определяет значение, возвращаемое функцией; выражение вычисляется, результат преобразуется к типу возвращаемого значения и передается в точку вызова функции;

- если функция не возвращает результата, оператор может, либо отсутствовать, либо присутствовать с пустым выражением: ***return*****;**

- функция может иметь несколько операторов ***return*** с различными выражениями, если алгоритм функции предусматривает разветвление.

Функция завершается, как только встречается оператор ***return***. Если функция не возвращает результата, и не имеет оператора ***return*****,** она завершается по окончанию тела функции.

Передача фактических параметров

Передача параметров может осуществляться тремя способами:

- **по значению**, когда в функцию передается числовое значение параметра;

- **по адресу**, когда в функцию передается адрес параметра, что особенно удобно для передачи в качестве параметров массивов;

- **по ссылке**, когда в функцию передается не числовое значение параметра, а сам параметр и тем самым обеспечивается доступ из тела функции к внешнему объекту.

Передача параметров по значению.

Формальным параметром может быть определение *скалярной переменной* или *структуры*, определенной пользователем. При вызове функции формальному параметру выделяется память в стеке, в соответствии с его типом. Фактическим параметром является – *выражение, значение* которого копируется в стек, в область памяти, выделенную под формальный параметр.

Фактическим параметром может быть просто неименованная константа нужного типа, или имя некоторой переменной, значение которой будет использовано как фактический параметр.

Все изменения, происходящие с формальным параметром в теле функции, не передаются переменной, значение которой являлось фактическим параметром функции.

Рассмотрим пример передачи параметров по значению:

```
… void  R (  int a, int b, int c)
   { a+=100;     b+=100;     c+=100;
    cout  << a<<b<<c<<endl ;}
int  main ( )
 { int a1=1,    b1=2,      c1=3;
   cout << a1<<b1<<c1<<endl ;
   R( a1, b1, c1 ) ;    // можно было вызвать так: R(1, 2 , 3),
   cout << a1<<b1<<c1<<endl;
  return 0; }
```

<u>Результат:</u>

1 2 3	// значение переменных до вызова функции
101 102 103	// значения формальных параметров
1 2 3	// значения переменных после вызова функции

Функция не изменила значения переменных, являющихся фактическими параметрами.

<u>Передача параметров по адресу - по указателю</u>

Формальным параметром является определение ***указателя type*p*** на переменную типа ***type*.** При вызове функции формальному параметру-указателю выделяется память в стеке 4 байта.

Фактическим параметром может быть либо адрес памяти переменной типа ***type*,** либо значение другого указателя, типа ***type**** из вызывающей программы.

В область памяти (в стеке), выделенную для указателя ***p*** будет копироваться значение некоторого *адреса* из вызывающей функции.

В теле функции, используя операцию разыменования указателя ****p*,** можно получить доступ к участку памяти, адрес которого получил ***p*** при вызове функции, и изменить содержимое этого участка**.**

Если в теле функции изменяется значение *p, при вызове функции эти изменения произойдут с тем объектом вызывающей программы, адрес которого использовался в качестве фактического параметра.

Рассмотрим пример передачи параметров по адресу:

```
. . .void  RR ( int* a,  int* b,  int*c)
    { *a+=100;     *b+=100 ;     *c+= 100 ;
      cout << *a << *b << *c; }
int main ( )
 { int a1 =1 , b1= 2 ,  c1=3 ;
```

```
  cout  << a1 << b1 <<c1  ;
  RR (  &a1 , &b1 , & c1 );            //передаем адреса переменных
  cout  << a1 << b1 <<c1  ;
  return 0; }
```

Результат:

1 2 3 // значения переменных до вызова функции
101 102 103 // изменения параметров в функции
101 102 103 // значения переменных после вызова

Функция изменяет внешние по отношению к ней переменные, используя механизм передачи параметра по адресу.

Передача параметров по ссылке

Ссылка определена в С++ как другое имя уже существующей переменной, которая и является инициализатором ссылки. Если в качестве формального параметра функции определена неинициализированная ***ссылка -*** *некоторая абстрактная переменная,* которая имеет имя, но не связана ни с каким участком оперативной памяти, в качестве фактического параметра при вызове функции следует использовать *имя переменной* из вызывающей программы того же типа, которая и свяжет ссылку со своим участком памяти.

Таким образом, ссылка обеспечивает доступ из функции непосредственно к внешнему участку памяти той переменной, которая при вызове функции будет использоваться в качестве фактического параметра.

Все изменения, происходящие в функции со ссылкой, будут происходить с переменной, являющейся фактическим параметром.

```
void  RRR (  int & a ,   int &b ,   int & c  )
 { a += 100;       b+= 100;   c+= 100;
   cout << a <<b <<c; }
int  main ( )
 { int a1=1 , b1 =2 ,   c1=3;
  cout <<  a1 << b1 <<c1 ;
  RRR( a1, b1 , c1 );
  cout << a1 <<b1 << c1 ;
  return 0;}
```

Результат:

1 2 3 // переменные до вызова функции
101 102 103 // изменения параметров в функции
101 102 103 // функция изменила фактические параметры

Функция непосредственно обрабатывает внешние по отношению к ней переменные. При этом в функции не выделяется память на формальные параметры, формальный параметр – это абстракция, имя, которое при передаче фактического параметра связывается с внешним по отношению к функции участком памяти.

Задача. Определить функцию, возвращающую произведение двух чисел, переданных в функцию посредством параметров. Функция должна возвращать результат, используя также формальный параметр (передача по адресу).

```
#include <iostream>
using namespace std;
void mult ( int a , int b , int *s )
{ * s = a *  b;  }
int  main ( )
{ int  c=3, d=7 , s;
  mult ( c, d , & s);  cout<<  s ;
 system("pause");     return 0; }
```

Задача. Определить функцию, которая находит минимальный и максимальный элементы в одномерном массиве – параметре функции и возвращает их посредством параметров – ссылок.

```
#include "stdafx.h"
#include <iostream>
#include <ctime>
#define n   100
using namespace std;
void minmax ( int a[n], int & min,  int & max )
 { min = max = a[0];
   for ( int i =0 ; i< n ; i++)
   if ( a[i] < min )  min = a[i]; else if ( a[i] > max ) max = a[i];  }
int main ( )
{ int b[n], min ,  max ;
 srand (time(0));            //инициализация генератора случайных чисел
 for ( int i=0 ; i< n ; i++)
 b[i] =rand() ;              //массив заполняем случайными числами
 minmax ( b , min ,  max );
 cout  <<  min  << max ;
system("pause"); return 0; }
```

Задача. Определить функцию, заполняющую трехмерный массив – параметр последовательными натуральными числами, начиная с единицы и возвращающую по ссылке сумму его элементов.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
const int n=4, k =5;
void f1 ( float a [ ][n][k] , int m, float & s )
{ s = 0; int t =1;
for ( int i=0; i< m; i++) for( int j=0; i< n; j++) for ( int l=0; l< k; l++)
{ a[i][j][l]=t++; s+= a[i][j][l];} }
int  main ( )
{ float s, dat [3][n][m];
f1 (dat, 3, s); //вызов функции заполняющей массив dat
cout << s; system("pause");
return 0;}
```

Задача. Определить функцию, формирующую упорядоченный массив из элементов двух других массивов. Массивы – параметры функции.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
void f2 (int*a, int*b, int*c, int n, int m)
{  int i, j, p;
   for ( i=0; i < n+m; i++)     //формирование неупорядоченного массива
      if (i < n) c[i] = a[i]; else c[i] = b[i-n];
  //упорядочивание массива методом пузырькового всплытия
   for (i = 0; i < n+m-1; i++) for (j = n+m - 1; j > i; j -- )
        if (c[j] < c[j-1]) {p= c[j]; c[j] = c[j-1]; c[j-1] = p;} }
int i, a [ ] = {7,9,5,4,0,2,89,33,73,11}, b [ ] = { 23,87,55,45,4,3,0,6,7,3},
n= sizeof( a ) / sizeof ( a[0]),
m= sizeof( b ) / sizeof ( b[0]);
int  main ( )
{ int*x = new int [n+m];   // - выделена память на динамический массив
 f2 (a , b , x , n , m ) ;
for ( i =0; i < n+m; i++)
cout << x[i] << " ";
system("pause");
delete [ ] x; return 0;}
```

Задача. Определить функцию, возвращающую с помощью ***return*** двумерный динамический массив натуральных чисел, размеры массива передать в функцию посредством параметров.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int** mas (int m, int n) { int k=1;
int ** ptr= new int* [m ];        //выделение памяти на массив из m указателей на int;
for (int i=0; i < m; i++) ptr [i] = new int [ n];   //выделение памяти, для каждой строки
for (int i =0; i < m; i++)       //заполнение массива и вывод его элементов;
{cout<<"\n";
for (int j = 0; j < n; j++)
 { ptr[ i] [j] = k++ ;cout<<ptr [i][j]<<" ";} }
return ptr; }     // возвращается "указатель на матрицу"
int  main ( )
{int n , m ,i ,j;
cin>> n>>m;    // размеры массива вводятся с клавиатуры;
int ** Q = mas (m, n );    //указателю присваивается результат вызова функции;
system("pause");
for (i=0; i < m; i++)   delete[ ] Q[i];delete [ ] Q; //освобождение памяти
return 0;}
```

1.3. Задания для самостоятельного решения

1) Определить функцию, возвращающую первое число в произвольном массиве кратное 7, если такие числа отсутствуют, функция должна возвращать 0. Возвращать результат, используя формальный параметр.

2) Определить функцию, результат работы которой двумерный массив, сформированный из произведений элементов каждого столбца некоторого массива С, находящихся в пределах: A< C[i][j] < B (первая строка нового массива) и количества таких элементов в каждом столбце (вторая строка нового массива). Массив С и граничные A и B – параметры функции.

3) Определить функцию, имеющую в качестве параметра трехмерный целочисленный массив. Результатом работы функции, является одномерный массив, возвращаемый с помощью **return**. Массив составляется из всех элементов трехмерного массива, кратных 3. На формируемый массив в функции выделяется динамическая память.

4) Определить функцию, возвращающую массив, составленный из четных элементов произвольного символьного массива – параметра функции (четные номера индексов).

5) Определить функцию, формирующую новый массив из двумерного массива вещественных чисел путем удаления из него строки и столбца, в которых находится минимальный элемент массива.

1.4. Контрольные вопросы

1) Дать определение функции. Синтаксис определения функции.
2) Что такое формальные и фактические параметры функции?
3) Оператор ***return***.
4) Передача параметров по значению, адресу и ссылке.
5) Массив (статический и динамический) – параметр функции.
6) Динамический массив – возвращаемый с помощью ***return.***

2. УКАЗАТЕЛЬ НА ФУНКЦИЮ – ПАРАМЕТР ФУНКЦИИ (занятие 8)

2.1. Цель занятия

Формирование компетенции (ОПК-5.3.4), получая навыки работы с библиотеками функций и классов. Формирование компетенций (ОПК 2.1.6, ПК-1.1.5), проводя разработки и отладки программ, используя современные системы программирования. Формирование компетенции (ПК-2.2.3), получая навыки работы с многомерными статическими и динамическими массивами.

2.2. Методические указания по теме

Указатель на функцию – это некоторая переменная, значениями которой являются адреса функций, характеристики которых (тип результата и сигнатура параметров) совпадают с характеристиками, указанными в определении указателя.

Имя функции - это ***указателем-константой*** на эту функцию, значением которого является адрес размещения операторов функции в оперативной памяти. Значение адреса можно присвоить ***указателю-переменной на функцию*** с тем же типом результата и с той же сигнатурой параметров. Указатель на функцию, инициализированный адресом некоторой функции

можно использовать для вызова этой функции. Определение указателя на функцию:

<тип_рез_ф> (* имя указателя) (спецификация_парам.) =
<имя инициирующей функции>;

- при определении достаточно перечислить через запятую типы параметров, имена параметров можно опустить;
- ***тип_рез_ф*** – это тип результата, возвращаемого функцией;
- инициализация указателя не обязательна, но при ее наличии тип результата, и сигнатура параметров инициирующей функции должны быть такими, же, как в определении указателя.

Эквивалентные вызовы функции с помощью указателя на эту функцию:

имя указателя (список фактических параметров);

(*имя указателя) (список фактических параметров);

Задача. Определить функцию вычисления длины строки **-** параметра функции (количества символов в строке до байтового нуля) В главной функции вызвать определенную функцию через указатель на функцию.

```
#include <iostream>
using namespace std;
int len (char* e) {
int m=0;
while (e[m++]);
return (m-1);}
int  main (){
int (*ptr) (char*);      //объявлен указатель на функцию
ptr = len;               //указателю присвоено значение – имя функции len;
char s [ ] = "rtgcerygw";
int n = ptr(s); cout<<n;
system("pause");return 0;}
```

Массивы указателей на функции.

Указатели на функции могут быть объединены в массивы. определение массива указателей на функции, возвращающие значение типа ***float*** и имеющие два параметра типа ***char*** и ***int***. В массиве с именем ***ptrArray*** четыре таких указателя:

float (*ptrArray) (char, int) [4]; или ***float (*ptrArray [4]) (char, int);***

При объявлении массива можно провести инициализацию элементов.

float v1 (char s, int n) {…}
…
float v4 (char s, int n) {…}
float (*ptrr [4]) (char, int) = { v1, v2, v3, v4 };

В рассматриваемом примере даны определения четырех однотипных функций ***v1,…v4*** и определен массив ***ptrr*** из четырех указателей, которые инициализированы именами функций ***v1,…v4.***

Чтобы вызвать, например, третью из этих функции, можно использовать такие операторы:

float x = (*ptrr [2]) ('a' , 5); или ***float x = ptrr [2] ('a' , 5);***

Для удобства последующих применений целесообразно вводить имя типа указателя на функцию с помощью спецификатора ***typedef:***

typedef <тип_функции> (*имя типа_указателя) (спецификация_параметров);

Массивы указателей на функции удобно использовать при разработке программ, управление которыми выполняется с помощью меню, реализующих вызов различных функций обработки данных в интерактивном режиме.

Рассмотрим алгоритм программы простейшего меню:

- варианты обработки определяются в виде функций ***art1() – act4();***
- объявляется тип ***menu*** **-** тип указателя на такие функции;
- объявляется массив ***act*** из четырех указателей на функции, инициированный именами функций ***art1() – act4();***

Интерактивная часть:

- на экран выводятся строки описания вариантов обработки данных и соответствующие вариантам целочисленные номера;
- пользователю предлагается выбрать из меню нужный ему пункт и ввести значение номера, соответствующее требуемому варианту обработки;
- пользователь вводит значение номера с клавиатуры;
- по номеру пункта, как по индексу, из массива указателей выбирается соответствующий элемент, инициированный адресом нужной функции обработки; производится вызов функции.

Использование массива указателей существенно упрощает программу, так как в данном случае отпадает необходимость использовать оператор ***switch*** – для выбора варианта. Ниже приведена программа:

```
# include < iostream>
# include < stdlib.h>
using namespace std;
// определение функций обработки данных:
void act1 ( ) { cout << "чтение файла"; }
void act2 ( ) { cout << "модификация файла"; }
void act3 ( ) { cout << "дополнение  файла"; }
void act4( ) { cout << "удаление записей  файла"; }
typedef void ( * menu) ();   // определение типа указателя на функции;
menu act [4] = { act1, act2 , act3 , act 4};          //определение массива указателей;
int  main () { int n;
cout << "\n1 - чтение файла";
cout << "\n2 - модификация файла";
cout << "\n3 - дополнение файла";
cout << "\n4 - удаление записей файла";
while (1) { cout << "\n введите номер"; cin >>n;
if ( n >= 1 && i<= 4) act [n-1] ( ); else exit(0);}
system("pause"); return 0;}
```

Указатель на функцию - параметр функции.

Указатели на функции удобно использовать в качестве параметров функций, когда объектами обработки должны служить другие функции.

В C++ все функции внешние, любую определенную функцию можно вызывать в теле любой другой функции непосредственно по имени, не передавая имя функции через механизм параметров.

Указатели на функции как параметры функций целесообразно использовать, когда в создаваемой функции должна быть заложена возможность обработки не конкретной, а произвольной функции.

В этом случае адрес обрабатываемой функции целесообразно передавать в функцию посредством параметра. В качестве формального параметра следует объявить указатель на функцию, а при вызове функции передавать в качестве фактического параметра адрес нужной обрабатываемой функции.

Указатели на функции в качестве формальных параметров можно, например, использовать в функциях:

- формирования таблиц результатов вычисления различных формул;
- вычисления интегралов с различными подынтегральными функциями;
- нахождения сумм рядов с различными общими членами и т. д.

Задача. Определить и вызвать функцию ***table()*** для построения таблицы значений различных формул.

Алгоритм задания:

- определяются три однотипных функции с одним вещественным параметром ***(a(x), b(x), c(x))*** для расчета значений, выводимых в таблицу;
- объявляется тип указателя ***func*** на такие функции;
- определяется массив ***S*** из трех указателей на функции инициированный именами функций ***a, b, c***;
- определяется функция ***table,*** выводящая в виде таблицы значения трех функций, передаваемых **в** ***table*** посредством параметров;
- аргументами функции ***table*** являются:

во-первых, массив указателей на функции с открытыми границами для передачи функций, вычисляющих значения и целочисленный параметр ***n*** для передачи количества таких функций;

во-вторых, параметры для аргумента функций – начальное значение ***xn***, конечное значение ***xk*** и шаг изменения аргумента ***dx*;**

- в главной функции производится вызов функции ***table()*** и передаются фактические параметры – массив ***S***, количество указателей в массиве - **3** и значения аргумента – начальное, конечное и шаг изменения аргумента.

Алгоритм функции table:

- устанавливается начальное значение аргумента функций ***x=xn*;**
- пока аргумент функций не достигнет своего конечного значения ***(x<= xk)*** выполняется повторяющаяся обработка: при каждом значении аргумента выводится строка значений трех функций, вызовы которых производятся с использованием указателей на функции из массива и затем значение аргумента увеличивается на величину ***dx***. Текст программы:

```
# include <iostream>
#include <cmath.>
using namespace std;
float a ( float x) { return x*x; }
float b ( float x) { return (x*x +100); }
float c ( float x) { return sqrt ( fabs(x)) +x;}
typedef float (* func) ( float x) ;
func S [3] = {a, b, c};
void  table ( func ptrA [ ] , int n, float xn , float xk , float dx )
{ float x = xn;
while ( x<= xk )
{ cout << "\n";  for (int i=0; i< n; i++) {cout. width(10); cout << (* ptrA[i] ) (x);}
 x+=dx ;} }
int  main () { table ( S, 3, 0., 2., 0.1 ); system("pause"); return 0;}
```

Задача. Определить функцию вычисления определенного интеграла с заданной точностью. Алгоритм:

- определяется подынтегральная функция с одним вещественным параметром ***f(x);***
- объявляется тип указателя ***func*** на функцию;
- определяется вспомогательная функция ***S*** вычисления площади под кривой подынтегральной функции, как суммы площадей прямоугольников, вписываемых в область под кривой. Такая сумма хорошо аппроксимирует функцию интеграла при достаточно большом разбиении ***n*** интервала аргумента ***[xn, xk]*** функции. Интервал определяют пределы интегрирования. Ширина прямоугольников равна (***xk- xn)/n***, высота равна ***f(x)*** при текущем значении аргумента. Аргумент изменяется от ***xn*** до ***xk*** с шагом (***xk- xn)/n.***
- функция интеграла будет определяться функцией ***S,*** когда ее значения с заданной точностью не будет зависеть от числа разбиений интервала интегрирования.

Текст программы:

```
# include <iostream>
#include <cmath.>
using namespace std;
float f ( float x) { return sqrt ( fabs( sin(x) +x);}//подынтегральная функция
typedef float (* func) ( float x); //тип указателя на функцию
float S (func fn, float xn, float xk, int n)
{ float x, dx=(xk-xn)/n, sum=0;
 for (  x=xn; x<=xk; x+=dx)
sum+=fn(x)*dx; return sum;}
float Integ (func fn, float a, float b, float e)//последний параметр точность
{ int n =20;                                   // зададим начальное число разбиений
 while( fabs( S(fn,a,b,2*n) - S(fn, a, b, n) )>e) n*=2;
 return S(fn, a, b, n); }
int main() { cout<<Integ ( f, 0.1, 30.1, 1.e-10); system("pause"); return 0;}
```

2.3. Задания для самостоятельного решения

1) Определить функцию нахождения суммы ряда с заданной точностью:

$$arctg(X)=\sum_{n=0}^{\infty}(-1)^n\cdot\frac{X^{2n+1}}{2n+1}=X-\frac{X^3}{3}+\frac{X^5}{5}-\frac{X^7}{7}+\ldots \text{ при } |X|<1;$$

Один из параметров функции – указатель на функцию, возвращающую значение очередного члена ряда, в зависимости от его номера в точке X.

2) Определить функцию, возвращающую значение определенного интеграла с заданной точностью. Вывести значение интеграла

$\int_a^b f(x)dx$ при $a=\frac{\pi}{2}$ *и* $b=\frac{3\pi}{2}$, *если* $f(x)=sin^2(x)$, *при* $0.0<x<\pi$

$f(x)=cos(x)$ *при* $\pi<x<2\pi$

с точностью 10^{-5}.

3) Определить функцию вывода на экран в ***m*** столбиков значения ***m*** однотипных функций с одним вещественным параметром. Использовать формальный параметр - массив указателей на функции. Вывести значения функций:

$$y1=x^2+1, y2=\sqrt{x^2+1}, y3=\sqrt{|x|+1}.$$

аргумент *x* меняется от 0 до 2 с шагом 0.1.

2.4.Контрольные вопросы

1) Указатели на функции (определение указателя, массива указателей, определение типа указателя на функцию).
2) Когда и как указатели на функции используются как параметры функций?
3) Что такое рекуррентная формула?

3. РЕАЛИЗАЦИЯ РЕКУРСИВНЫХ АЛГОРИТМОВ. (Занятие 9).

3.1. Цель занятия

Формирование компетенции (ОПК 1.1.3) в виде получения навыков тестирования и отладки приложений, поиска основных типов ошибок и их локализации. Формирование компетенции (ОПК-5.3.4), получая навыки работы с библиотеками функций и классов.

3.2. Методические указания по теме

Рекурсия – это способ организации обработки данных в функции, когда функция обращается сама к себе прямо или косвенно.

Рекурсивная форма алгоритма дает более компактный текст программы, но требует дополнительных затрат оперативной памяти для размещения данных и времени для рекурсивных вызовов функции.

Рекурсивный алгоритм позволяет повторять операторы тела функции многократно, каждый раз с новыми параметрами, конкурируя тем самым с итерационными методами (циклами).

И также, как и в итерационных методах, алгоритм должен включать условие для завершения повторения обработки данных, то есть иметь ветвь решения задачи без рекурсивного вызова функции.

Использование рекурсии не дает никакого практического выигрыша в программной реализации, и ее следует избегать, когда есть очевидное итерационное решение.

Задача. Определить рекурсивную функцию, возвращающую значение факториала целого числа. Факториал определятся только для положительных чисел формулой:

$N! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot ... \cdot N$, и факториал нуля равен *1* ***(0! =1).***

Определение факториала можно переписать в виде "математической рекурсии":

$N! = N \cdot (N-1)!$**,** то есть факториал вычисляется через факториал.

Соответственно определение функции имеет вид:

```
long fact ( int k)
{if ( k <0 ) return 0 ;  //полагаем, что факториал в отрицательной области равен 0
if ( k = = 0 ) return 1;                    // здесь рекурсия прерывается;
return k * fact ( k- 1);}
```

Задача. Определить рекурсивную функцию, возвращающую целую степень вещественного числа. Определим сначала "математическую рекурсию". Определение *n*-ой степени числа *X* можно представить в виде:

$X^n = X * X^{n-1}$ *при n >0,*

$X^n = X^{n+1} / X$ *при n< 0,*

и в соответствии с этим определить рекурсивную функцию:

```
double step ( double X , int n )
{ if (n = = 0) return 1;          // здесь рекурсия прерывается;
if (X = = 0) return 0;
if (n > 0 ) return X * step(X , n-1 );
if (n < 0 ) return step(X , n+1 ) / X;
}
```

Задача. Определить функцию, возвращающую сумму элементов одномерного массива. Запишем формулу для суммы *n* элементов в виде суммы последнего элемента и суммы первых *n-1* элементов**:**

$S_n = a[n-1] + S_{n-1}$**,** то есть сумма вычисляется через сумму.

Соответственное определение рекурсивной функции:

```
int sum (int a[] , int n )
{ if (n = = 1) return ( a[0] ); // здесь рекурсия прерывается;
else return ( a[n-1] + sum (a, n-1) );}
```

3.3. Задания для самостоятельного решения

1) Определить рекурсивную функцию для многократного вывода символьной строки. Количество выводов и выводимая строка – параметры функции. В главной функции, вызвать функцию, 10 раз вывести на экран фразу.

2) Определить рекурсивную функцию, которая для произвольного ряда возвращает сумму **m** членов ряда, начиная с **n**-го члена ряда. Функция, имеет один из параметров – указатель на функцию, возвращающую значение члена ряда, в зависимости от его номера, в точке x. Вывести на экран значение суммы членов с 5–го по 25–й член, следующего ряда: $\sum_{n=1}^{\infty}\left(\frac{(n+1)!}{3^{n+1}}+\frac{1}{x^{n+1}}\right)$.

3) Определить рекурсивную функцию, которая для произвольного ряда возвращает значение сумму членов ряда с заданной точностью. Функция имеет один из параметров – указатель на функцию, возвращающую значение члена ряда, в зависимости от его номера, в точке x. Вывести на экран значение суммы ряда $\sum_{n=1}^{\infty}(-1)^{n+1}\frac{2^{2n-1}\cdot X^{2n}}{(2n)!}$ с точностью 10^{-5}.

4) Определить рекурсивную функцию для вычисления следующей математической функции *F(n)* (аналог факториала) , где *n* – целое :

$F(n) = 3\cdot 6\cdot 9\cdot\ldots\cdot(3\cdot n)$, при $n>=1$ и $F(n) = 0$, при $n<1$.

5) Определить рекурсивную функцию для вычисления следующей математической функции *F(n)* (аналог факториала), где *n* – целое:

$F(n) = 1\cdot 3\cdot 5\cdot\ldots\cdot(2\cdot n-1)$; при $n>=1$ и $F(n) = 0$, при $n<1$.

3.4. Контрольные вопросы

1) Что такое рекурсивная функция, какова ее структура?
2) Рекурсивный алгоритм – альтернатива циклического алгоритма.
3) Прямая и косвенная рекурсия.
4) Достоинства и недостатки рекурсивного алгоритма.
5) В каких задачах целесообразно использовать рекурсивный алгоритм.
6) Что такое рекуррентная формула.
7) Преимущества использования рекуррентной формулы

4. РАЗРАБОТКА АЛГОРИТМОВ И ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ СТРУКТУР И МАССИВОВ СТРУКТУР. ДИНАМИЧЕСКОЕ ВЫДЕЛЕНИЕ ПАМЯТИ (Занятие 10)

4.1. Цель занятия

Формирование компетенции (ОПК 1.1.3) в виде получения навыков тестирования и отладки приложений, поиска основных типов ошибок и их локализации. Формирование компетенции (ОПК-5.3.4), получая навыки работы с библиотеками функций и классов. Формирование компетенций (ОПК 2.1.6, ПК-1.1.5), проводя разработки и отладки программ, используя современные системы программирования. Формирование компетенции (ПК-2.2.3) в виде получения навыков работы с многомерными статическими и динамическими массивами.

4.2. Методические указания по теме

Структура - это объединенное в единое целое множество поименованных элементов (полей) в общем случае разных типов.

Элементы структуры могут быть разных типов и все они должны иметь различные имена.

Перед определением ***структуры*** надо определить ее ***структурный тип***, который и задает внутреннее строение структуры.

<u>Простейший формат определения структурного типа:</u>

struct имя_типа
{список_компонентов};

Определение структурного типа, всегда заканчивается **';'.**

Список компонентов – это определения и описания, типизированных данных (полей), разделенных **';'.**

Например, определим структуру, описывающую характеристики человека:

- имя человека (строка символов);
- возраст человека (целое число).

Определение типа такой структуры**:**

struct men {char name [30]; int age; };

После определения типа можно определять конкретные структуры этого типа, массивы структур, указатели на структуру:

men men1, men2, mens [10], * m;
men1, men2 – две структуры типа ***men***;
mens **-** имя массива из 10 структур типа ***men***;
m **-** указатель на структуру типа ***men***;

При определении структурного типа не происходит выделения памяти!

Выделение памяти происходит при определении переменных данного структурного типа, то есть при определении ***структур.*** Элементы структуры располагаются в памяти подряд. Количество памяти на каждый элемент выделяется в соответствии с его типом элемента. Количество памяти, выделенное под структуру, определяется суммой байт участков, выделенных под ее элементы.

Если нет необходимости объявлять в разных частях программы структуры одного типа, можно не вводить именованный тип, а сразу в строке определения типа объявить все необходимые объекты, например,

struct { char fio[14]; int nz; float st; } A, B , Stud [10] , * Ptr ;

<u>Для обращения к элементу структуры чаще всего используется уточненное имя:</u>

имя_ структуры. имя_элемента_структуры

Например, допустимо использовать следующие операторы:

cin>>A. fio; *// ввод значения в поле **fio** структуры **A***
cout<<A.fio; *// вывод значения элемента*
cin>>Stud [5]. fio; *// ввод значение компонента **fio** элемента массива **Stud***
B.st = 2590.5; *// полю **st** структуры **B** присваивается значение*
cout<<B.st *// вывод значения поля*

Символьные данные можно представлять двумя способами:

1) в виде символьного массива: ***char fio [14];***

В этом случае имени массива нельзя присваивать строку, так как имя массива – указатель константа и присвоить ему новый адрес строковой константы нельзя. То есть, оператор: ***A.fio = "Петров"; –*** не допустим!

Чтобы поместить в это поле строку с фамилией надо воспользоваться операцией копирования: ***strcpy (A.fio, "Петров");***
Функция ***char* strcpy (char*s1 , char* s2);*** описанная в модуле ***cstring.h,*** копирует символы строки ***s2*** в строку ***s1*** и возвращает строку ***s1.***

Также допустимо введения значения компонента с клавиатуры, так как память под массив ***fio*** выделена: оператор ***cin>>A.fio;*** - допустим.

2) Фамилию в структуре можно представлять, используя указатель: ***char* FIO;*** и тогда допустима операция присваивания указателю адреса строки:
A.FIO ="Григорьев";

При объявлении указателя в качестве компонента структуры для представления массива символов (например, – фамилии студента) выделяется память только на указатель, а на элементы массива память не выделяется.

Если в данном случае надо ввести значение строки, например, с клавиатуры или из файла, следует выделить участок памяти и присвоить указателю ***FIO*** адрес участка**.** Затем произвести ввод строки:
A.FIO=new char [14];
cin>>A.FIO;

При определении структуры можно провести её инициализацию – задание начальных значений её элементов.

В этом случае после определения структуры ставится знак *'='* и следует список инициализации, заключенный в фигурные скобки, в котором через запятую перечисляются начальные значения элементов структуры.

Определим структурный тип:

struct student {char * FIO; int nz; int maks [3]; float ball;};

Определим структуру типа ***student*** с инициализацией:
student one =
{"Петров", 4123, {4, 3, 2}, (one.maks [0] +one.maks [1] +one.maks [2])/3.0};

Можно определять указатели на структуры:

имя_структурного_типа * имя_ указателя_ на_ структуру = инициирующее_выражение

Пример:

student * ptr = &one; //указатель инициирован адресом структуры ***one***

После того как указатель получил в качестве значения адрес структуры, обращаться к элементам структуры, можно используя указатель:

1) ***имя_указателя -> имя элемента структуры***

cout<<ptr ->FIO ;

2) разыменование указателя и формирование уточненного имени:

(*имя_указателя). имя_элемента_структуры

cout<< (*ptr). FIO;

Компонентами (членами) структура могут быть объекты любых типов - скалярных, так и типов, определенных пользователем.

Единственное существенное ограничение - элементом структуры не может быть структура или массив структур того же типа.

В то же время элементом определяемой структуры может быть указатель на структуру определяемого типа.

Вложенные структур.

Элементом структуры может быть другая структура, тип которой уже определен. Такая структура называется вложенной и к ее полям следует обращаться с двойным уточнением.

имя_внешней_структуры.имя_вложенной_структуры.имя поля;

Динамическое выделение памяти

Для выделения памяти используется операцию ***new***. Для освобождения – операция ***delete***. Определим структурный тип – ***book*** с данными о книге:

struct book { char * name, * author; int year, pages; };

//Далее надо объявить указатель на структуру:

book * une;

une = new (book); *// выделение памяти на одну структуру*

delete une;// освобождение памяти

une = new book [9];*// выделение памяти на массив структур*

une[0].name = "Денисов";

delete [ ] une ;*//освобождение памяти*

Структуры и функции.

Через аппарат формальных параметров информация о структуре или массиве структур может передаваться в функцию по значению, по адресу и по ссылке. Функции могут также возвращать структурные данные (отдельные структуры или массивы структур) как результат с помощью оператора ***return***.

Задача. На курсе три группы. Данные о группах курса должны включать:

- название группы;
- количество студентов в группе;
- массив данных по каждому студенту группы включает:

- ФИО;
- номер зачетки;
- стипендия;

Задание:

1) определить функцию ввода данных с клавиатуры, по ссылке возвращающую суммарную стипендию;
2) определить функцию форматного вывода данных в таблицу, в алфавитном порядке:

Группа :

ФИО студента	Номер зачетки	Стипендия
50	10	8

Текст программы:

```
#include "stdafx.h"
#include <iostream>
#include <cstring>
#include <iomanip>
using namespace std;
struct stud {char fio[50]; unsigned  int nz; float rs;};
struct grup { char name[10]; unsigned int n; stud*M;} GR [3];
char t[80];
char*sh[5 ]= {"… ",",…",",…",",…",",…"};   //строки таблицы
//--------------------убирает пробелы ведущие и конечные в строке- параметре---------
void filtr(char* stroka){
char S[255];
int xl=0, // номер первого символа строки
xr= strlen(stroka)-1;//номер последнего символа строка
while(stroka[xl]==’  ‘)  xl++;
while(stroka[xr]==’  ‘ ) xr--;
for( int x= xl; x<=xr ; x++)  S[x-xl)=stroke[x];
S[x-xl)=”\0’;strcpy(stroke,S); }
//------------------------------вводит данные одной группы---------------------------------------
void vvod (grup&G, float&stip)
{stip=0;
cout<<”Введите название группы:”; cin.getline(G.name,10); filtr(G.name);
cout<<<<”Введите кол-во студентов:”;cin>>G.n;
G.M=new stud [G.n]; //выделили память на массив студентов
for(int i=0; i<G.n; i++)
{cin.getline(t,80);
cout<<”Введите ФИО студента:”; cin.getline (G.M[i].fio,50); filtr(G.M[i].fio);
cout<<”Введите номер зачетки:”; cin>>G.M[i].nz;
cout<<”Введите стипендию:”; cin>>G.M[i].rs; stip+= G.M[i].rs;}
}
//-----------------------------------сортирует одну группу--------------------------------------
void sort ((grup&G){
for(int i=0; i<(G.n)-1; i++)  for(int j=i+1; j<G.n; j++)
```

```
if(strcmp(G.M[i].fio,G.M[j].fio)>0)
{st=G.M[i]; G.M[i]=G.M[j]; G.M[j]=st;} }
//----------------------------------выводит данные одной группы---------------------------------
void vivod ((grup&G){
cout<<"\n              Группа: "<<G.name<<endl;
for(int i=0; i<5; i++)  cout<<sh[i]<<endl;
for(int i=0; i<G.n; i++)
cout<<"|"<<setw(50)<<G.M[i].fio<<"|"<<setw(10)<<  G.M[i].nz<<"|"<<setw(8)<<
G.M[i].rs<<"|"<<endl;
cout<<sh[5]<<endl;   }
//-----------------------------------------------------------------------------------------------
int main() {
float sum;
for (int i=0; i<3; i++)
{ vvod(GR[i],sum); sort(GR[i]); vivod(GR[i]);
cout<"Суммарная стипендия группы: "<<sum<<endl;}
system("pause"); return 0;}
```

4.3. Задания для самостоятельного решения

1) Имеются три каталога книг по предметам: физика, математика, биология. Данные этих каталогов включают:

- наименование предмета;
- количество книг по предмету;
- данные по каждой книге:
    - название книги;
    - автор;
    - год издания;
    - издательство;

Задание:

- Определить функцию ввода данных с клавиатуры.
- Определить функцию, форматного вывода данных каждый каталога, отсортированных по названию книг, и возвращающую количество книг каталога.

Предмет :

Название	Автор	Год издания	Издательство
50	25	**5**	**12**

- Ввести и вывести данные каталогов.

2) Имеются данные о 2000 междугородних телефонных разговорах, загруженные в массив структур. Данные включают:

- ФИО абонента;
- номер телефона;
- адрес абонента

- дата телефонного разговора ;
- время в минутах;
- индекс региона;
- коэффициент региона

Определить функцию, распечатывающую счета за междугородние разговоры для каждого абонента форматно в таблицу:

Адрес абонента:

ФИО абонента	**Номер** абонента	**Дата разговора**			**Индекс** региона	**Время** Разговора (в минутах)	**Плата за разговор** (в руб. и коп.)
		день	месяц	год			
20	10	4	4	4	7	4	8

Суммарная плата:

Плата за разговор рассчитывается по формуле: коэффициент региона * 100*время в минутах.

4.4. Контрольные вопросы

1) Что такое структура и как ее определить?
2) Инициализация структуры, массива структур.
3) Какой объем оперативной памяти получает структура, массив структур?
4) Динамическое выделение памяти на структуру, массив структур.
5) Как можно обращаться к полям структуры или элемента массива структур, если определены структура и массив структур?
6) Как можно обращаться к полям структуры или элемента массива структур, если объявлен указатель на структуру или указатель на элемент массива структур?
7) Структура – параметр функции (передача параметра по значению, по адресу, по ссылке).

5. СОЗДАНИЕ СВЯЗАННЫХ СТРУКТУР - СПИСКОВ ДЛЯ РЕШЕНИЯ РАЗЛИЧНЫХ ПРАКТИЧЕСКИХ ЗАДАЧ (Занятие 11)

5.1. Цель занятия

Формирование компетенции (ОПК 1.1.3) в виде получения навыков тестирования и отладки приложений, поиска основных типов ошибок и их локализации. Формирование компетенции (ОПК-5.3.4), получая навыки работы с библиотеками функций и классов. Формирование компетенций (ОПК 2.1.6, ПК-1.1.5), проводя разработки и отладки программ, используя современные системы программирования. Формирование компетенции (ПК-2.2.3) в виде получения навыков работы с многомерными статическими и динамическими массивами и структурами данных.

5.2. Методические указания по теме

Связанные структуры

Если программа должна работать с фиксированным числом экземпляров одного типа структуры, то объявляется массив структур.

Если

- количество экземпляров не известно – статический массив для работы с данными не подходит;

- количество экземпляров структур определяется только в конце работы программы – динамический массив также не подходит.

Таким образом, если заранее не известно, сколько будет структур, можно пользоваться связанными структурами, в которых каждая отдельная структура связана с помощью указателей с соседней или с соседними.

Рассмотрим линейные однонаправленные связанные структуры.

1) Все элементы связанных структур представляют собой последовательность элементов, связанных в цепь посредством указателей.

2) Память для элементов выделяется и освобождается динамическая (***new – delete*** *или* ***malloc – free*** *)*

3) Последовательность элементов имеет вершину (начало) и хвост (конец).

4) Структуры последовательности связываются только с одной соседней структурой – предыдущей или последующей.

К линейным связанным структурам относятся стеки, очереди, списки, включение и исключение элементов в которых происходит по-разному:

- в **стеке** включение и исключение происходит только с одного конца – с вершины стека;

- в **очереди** включение структуры происходит в хвост цепи (конец очереди), а исключение происходит с вершины цепи (начало очереди);

- в **списке** включение и исключение элементов происходит по-разному, например, если элементы списка упорядочены по определенному признаку (одному из полей структуры), включение и исключение происходит в любом месте цепи, так чтобы не нарушать порядок списка.

Элемент связанной структуры должен состоять из данного в обычном понимании и из ссылочного элемента (указателя), посредством которого будет осуществляться связь структуры с другими структурами.

Список

Рассмотрим сортированный список, в котором элементы упорядочены по убывания или возрастания ключевого признака – одного из полей структур.

Рассмотрим следующие функции:

1) добавление элемента в список (рис.4);
2) удаление элемента с заданным значением поискового признака;
3) поиск и вывод заданного элемента;
4) чтение и вывод всех элементов;
5) уничтожение списка путем освобождения ОП

Будем полагать, что данными структур списка являются структуры, описывающие некоторые характеристики книг.

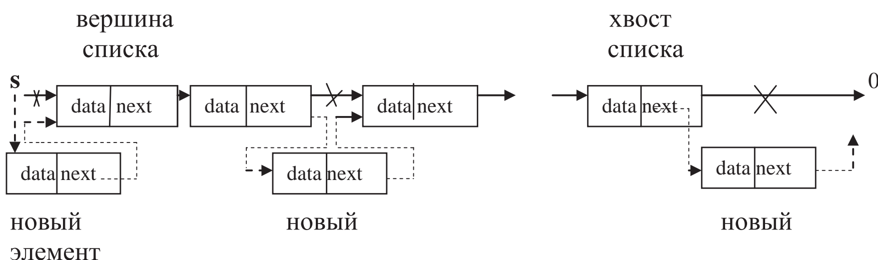

Рисунок 4. Схема добавление структуры в список

Программная реализация списка.

```
#include<cstring>
#include<iostream>
using namespace std;
int count ;  // внешняя переменная для нумерации структур в списке
// описание структурного типа узла списка:
struct spisoc{
char*name;     // название книги
char*author;   // имя автора
int year;           // год издания
spisoc* next;    // указатель на следующую структуру в списке
};
// определение массива структур для дополнения с инициализацией
spisoc stm[ ]={   { "Программирование на С++", "Страуструп Б.", 2013},
                        { "Turbo C++" , "Винер Р.С.", 1991 },
                       { "С-С++", "Березин Б.И. " , 2006 } };
spisoc *s;     // определение внешнего указателя на вершину списка,
                       // он по - умолчанию инициализирован нулем (null)
//---------------------------------------------------------------------------
// Функция вывода  данных одной структуры – параметра функции
void print (spisoc st){
cout<<"\n"<<++count<<". "<<st.name<<", " <<st.author<<", " <<st.year;}
// Функция вывода без нумерации
void print1(spisoc st)
{ cout<<"\n"<<st.name<<", " <<st.author<<", " <<st.year;}
//---------------------------------------------------------------------
// Функция дополнения списка структурой, передаваемой через параметр,
// адрес вершины списка передается по ссылке
void dop (spisoc*&s, spisoc*st) {
spisoc*list = s, // текущий указатель инициализируется указателем  на
                       //вершину
*pr=0,              // вспомогательный указатель инициализирован 0
*stnew=new(spisoc);       // выделяется память на новую структуру
                                   // типа   spisoc
```

```
*stnew=*st;          // копируются данные параметра в новую структуру
// Организуем цикл продвижения по списку пока не достигнем  нулевого указателя
или пока не найдем в списке такую структуру, в которой поле названия
// в алфавитном порядке располагается после поля названия вводимой структуры
while(list && (cstrcmp (list->name, st->name) <= 0))
{ pr=list;     // pr присваиваем значение указателя на
              // текущую структуры, а
 list=list->next;  //текущему указателю list присваиваем значение
 //указателя на следующую структуру list->next
}
stnew->next=list;            //производим подключение справа: новую
                               // запись помещаем перед  list
// организуем подключение слева
if(pr= =0) s=stnew;    // если не только list но и pr равен нулю, это
                          // значит, что список пуст и поэтому вершину
                          //надо установить на новый элемент
else pr->next=stnew; } // подключение слева
//-----------------------------------------------------------------------------------------------
//Функция чтения списка, использующая цикл для продвижения по списку,
// адрес вершины списка передаем по значению
void cht (spisoc*s){
cout<<"\n\nСтруктуры списка :";
count=0;     //  устанавливаем нумерацию структур на 0
spisoc*list=s;        // установка текущего указателя на вершину списка
if (list = = 0) cout<<"\nСписок пуст"<<endl;
else  while(list)      // пока list отличен от нуля
 { print( *list);        // выводим структуру
list=list->next;   // переходим к следующей структуре
 }
count=0; }        //внешняя переменная - нумерация структур обнуляется
//-------------------------------------------------------------------------------
//Функция поиска структуры первой попавшейся в списке, у которой
//поле - год издания совпадает с параметром,
// адрес вершины списка передаем по значению
void poisk (spisoc*s, int year)
{ spisoc*list=s;  // установка текущего указателя на вершину
If (list==0) cout<<"\nСписок пуст"<<endl;
else {
while (list&&list->year!=year) //пока list не достиг нуля и  поле «год издания» текущей
структуры не совпадает  со значением параметра
list=list->next;      // продвигаемся по списку
//если цикл завершился, при достижении нуля - искомой структуры нет в списке,
//в противном случае, структура найдена,  выводим эту структуру
If (list==0) cout<<"\nЭлемент не найден"<<endl;
else {cout<<"\n\n Результат поиска :";print1 (*list);}
     } }
//-----------------------------------------------------------------------------
//Функция удаления одной первой попавшейся структуры, у
//которой значение поля – год издания совпало со значением параметра функции,
```

```
// адрес вершины списка передается по ссылке
void ud (spisoc*&s, int year)
{ spisoc*list=s,  // устанавливаем текущий указатель на вершину списка
 *pr=0;             // вспомогательный указатель инициализируется нулем
// организуется цикл продвижения по списку
while( list&&list->year!=year) //пока list не достиг нуля и не найдена структура,
              //у которой поле year совпадает со значением параметра
{  pr=list;              //pr присваиваем значение текущего указатель,
  list=list->next; }   //текущему указателю присваиваем адрес следующей структуры
  if (list= =0) cout<<"\nЭлемент не найден"; // если цикл прервался по
                                                      //достижению конца списка
else{  cout<<"\n\n“Удаление: "; print1(*list);    // в противном случае
                                              //выводится удаляемая структура
 // удаление структуры из списка
If (pr= =0)            // если найденная структура – первая в списке
s=list->next;         // указателю на вершину присваивается адрес следующей
                      // за list  структуры
 //в противном случае – посредством указателей предыдущая структура от
//найденной связывается со следующей за ней
else pr->next=list->next;
delete(list);      // память  освобождается
} }
//---------------------------------------------------------------------------------------------------------
//Функция уничтожения списка
void osv (spisoc*&s) {
 spisoc*list=s, // текущий указатель устанавливается на вершину списка
*pr=0;              // вспомогательный на null
while(list) {       // пока не достигнем конца списка
pr=list;              //значение текущего указателя запоминаем в pr
list=list->next;  //текущему указателю присваиваем адрес  следующей структуры
delete(pr); }          // последовательно освобождаем  память
 s=0;}                 // указатель на вершину получает значение null;
int main() {
for(int i=0;i<3; i++) dop(s,stm+i);
cht(s);
poisk(s,1998);
cout<<"\n\nУдаление:";
ud (s,1991);
cout<<"\n\nСтруктуры списка :";
cht(s); osv(s); cht(s);
system("pause");
return 0;}
```

Схема двусвязной линейной структуры представлена на рис. 5.

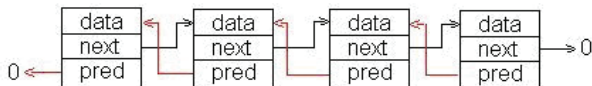

Рисунок 5. Двусвязный список

5.3. Задания для самостоятельного решения

1) Определить функцию дополнения списка данными вводимыми, а) с клавиатуры до нажатия символа ***Esc***, б) из файла.

2) Определить рекурсивную функцию чтения списка, при этом данные выводить в некоторый текстовой файл.

3) Определить рекурсивную функцию удаления всех структур списка, у которых значение поля данных совпало с ключевым значением.

4) Определить функцию поиска всех структур списка, у которых значение поля данных совпало с ключевым значением.

5.4. Контрольные вопросы

1) Дать определение связанных динамических структур.
2) Односвязные, двусвязные линейные структуры.
3) Дать примеры односвязных динамических структур, основные отличия.
4) Назначение, преимущества списков.

6. РАЗРАБОТКА СВЯЗАННЫХ СТРУКТУР - ОЧЕРЕДЕЙ И ФУНКЦИЙ РАБОТЫ С НИМИ (Занятие 12)

6.1. Цель занятия

Формирование компетенции (ОПК 1.1.3) в виде получения навыков тестирования и отладки приложений, поиска основных типов ошибок и их локализации. Формирование компетенций (ОПК 2.1.6, ПК-1.1.5), проводя разработки и отладки программ, используя современные системы программирования (VS). Формирование компетенции (ПК-2.2.3) в виде получения навыков работы с многомерными статическими и динамическими массивами и структурами данных.

6.2. Методические указания по теме

Очередь

Это связанная динамическая структура данных, в которой добавлять элементы можно только в конец (хвост) очереди, а удалять и читать можно только элементы с начала (вершина) очереди (рис.6).

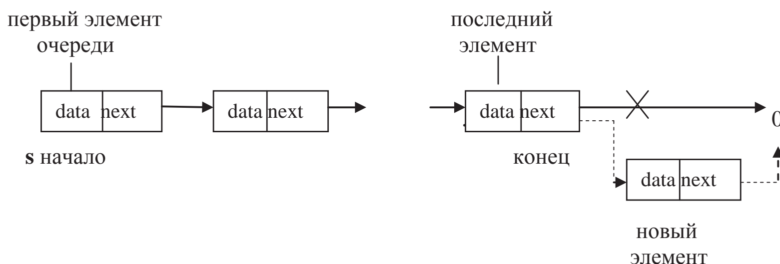

Рисунок 6. Односвязная очередь, добавление нового элемента.

Программная реализация очереди.

```
#include <iostream>
using namespace std;
struct qu {  int data ;  qu* next ; };      //узел
//-----------------------------------------------------------------------------------------------
void dop ( qu *&s,   int dat) // ссылка на указатель на начало  очереди
{ qu * p = s,         //текущий указатель устанавливаем на начало очереди
        *q = 0,       // вспомогательный указатель - на null
      *nst=  new ( qu); // выделение память для нового элемента
nst ->data = dat; //присваивание данному новой структуры значения параметра
nst->next=null; // новый элемент -  последний элемент очереди
while ( p)         // пока указатель не достигнет нулевого значения
{ q=p ;  p= p ->next ; };   //продвигаемся по очереди к концу
if (q)    q ->next= nst ;   //если q не равно нулю, последний ненулевой элемент должен
                                       //   указывать на новый элемент
else s = nst; }         //в противном случае -  новый элемент первый в очереди
//-----------------------------------------------------------------------------
// Функция удаления элемента из очереди, возвращает удаляемую информацию.
//Удаление производится из вершины очереди
int ud ( qu*&s)
{    int k=0;
    qu* fr =s    //  указатель на начало очереди
     if(s)
{ k = s -> data ; s = s -> next ; delete fr;}
return k; }
//-----------------------------------------------------------------------------
// Функция чтения элемента из начала очереди
int cht ( qu*s )
{ if(s) return s ->data ; }
//----------------------------------------------------------------------
// Функция чтения элементов очереди
int cht1 ( qu*s ) {
qu* list=s;
if(list==NULL) cout<<"\nЭлементов нет";
else while(list)
      {cout<<"\n data = "<< list->data;
       list=list->next;}
  }
qu * s1;       // объявлен указатель на очередь глобальный
//-----------------------------------------------------------------------------
int  main ( ){
for (int i=1 ; i<=20 ; i++)
dop ( s1 , i );
cout<<cht (s1 );
cout<<ud(s1);
```

```
cout<<cht1 (s1 );
system("pause");
return 0; }
```

6.3. Задания для самостоятельного решения

1) Определить очередь и функции работы с ней для структурных данных, описывающих, характеристики студента (ФИО, номер группы, оценки сессии, средний балл).

2) Определить функцию дополнения очереди данными вводимыми, а) с клавиатуры до нажатия символа ***Esc***, б) из файла.

3) Определить рекурсивную функцию чтения очереди, при этом данные выводить в некоторый текстовой файл.

4) Определить рекурсивную функцию удаления всех структур очереди, у которых значение поля данных совпало с ключевым значением.

5) Определить функцию поиска всех структур очереди, у которых значение поля данных совпало с ключевым значением.

6.4. Контрольные вопросы

1) Дать определение связанных динамических структур.
2) Односвязные, двусвязные линейные структуры.
3) Дать примеры односвязных динамических структур, основные отличия.
4) Назначение, преимущества очереди.

7. АЛГОРИТМЫ СОРТИРОВКИ МАССИВОВ (Занятие 13)

7.1. Цель занятия

Формирование компетенции (ОПК 1.1.3) в виде получения навыков тестирования и отладки приложений, поиска основных типов ошибок и их локализации. Формирование компетенции (ОПК-5.3.4), получая навыки работы с библиотеками функций и классов. Формирование компетенций (ПК-2.1.9) в виде получения навыков работы с алгоритмами быстрого поиска и сортировки данных. Формирование компетенции (ПК-2.2.3) в виде получения навыков работы с многомерными статическими и динамическими массивами и структурами данных.

7.2. Методические указания по теме

Алгоритмом сортировки называется алгоритм для упорядочения некоторого множества элементов.

Ключом сортировки называется атрибут (или несколько атрибутов), по значению которого определяется порядок элементов, по возрастанию или убыванию ключа.

Таким образом, при написании алгоритмов сортировок массивов следует учесть, что ключ полностью или частично совпадает с сортируемыми данными.

Практически каждый алгоритм сортировки можно разбить на 3 части:

- *сравнение*, определяющее упорядоченность пары элементов;
- *перестановку*, меняющую местами пару элементов;
- собственно, *сортирующий алгоритм*, который осуществляет *сравнение* и *перестановку* элементов до тех пор, пока все *элементы множества* не будут упорядочены.

Оценка алгоритмов сортировки

Универсального, наилучшего алгоритма сортировки на данный момент не существует. Однако, имея приблизительные характеристики *входных данных*, можно подобрать метод, работающий оптимальным образом. Для этого необходимо знать параметры, по которым будет производиться оценка алгоритмов, такие как

- *Количество сравнений и пересылок данных.*
- *Время сортировки* – основной параметр, характеризующий быстродействие алгоритма.
- *Память* – один из параметров, который характеризуется тем, что ряд алгоритмов сортировки требуют выделения дополнительной памяти под временное хранение данных.
- *Устойчивость* – это параметр, который отвечает за то, что сортировка не меняет взаимного расположения равных элементов.
- *Естественность поведения* – параметр, которой указывает на эффективность метода при обработке уже отсортированных, или частично отсортированных данных.

Классификация алгоритмов сортировок

Алгоритмы сортировки подразделяются на *внутреннюю и внешнюю сортировки.*

Внутренняя сортировка – это алгоритм сортировки, который в процессе *упорядочивания данных* использует только *оперативную память* компьютера. То есть, сортируемый массив данных с произвольным доступом к любой ячейке помещен в оперативную память для выполнения алгоритма. В конкретных алгоритмах, может использоваться дополнительная оперативная память.

Внешняя сортировка – это алгоритм сортировки, который при проведении *упорядочивания данных* использует *внешнюю память*. *Внешняя сортировка* предполагает обработку больших массивов данных. Обращение к различным носителям накладывает некоторые дополнительные ограничения на алгоритм. *Внутренняя сортировка* является базовой для любого алгоритма *внешней сортировки*. Внутренняя сортировка значительно эффективней внешней.

На рис. 7 представлен ряд известных методов сортировки.

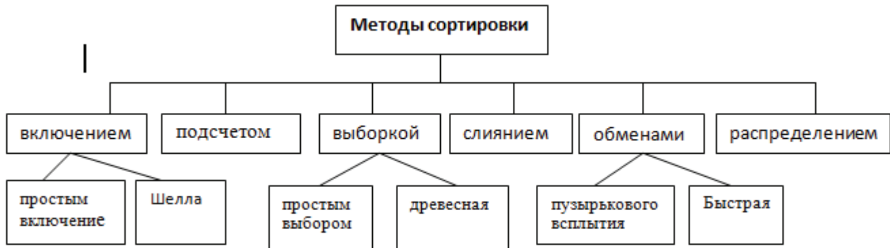

Рисунок 7. Алгоритмы внутренней сортировки.

Рассмотрим некоторые алгоритмы внутренних сортировок.

Начнем с методов простых для понимания и надежных для реализации: ***простого включения, простого выбора и пузырькового всплытия***. Все они сравнительно неэффективны, их среднее время исполнения пропорционально квадрату числа сортируемых элементов **O(n^2).** Когда **n** не велико, они могут оказаться достаточно эффективными. Из простых методов рассмотрим сортировку «простым включением» или как ее еще называют – «простыми вставками». Все рассматриваемые программы сортируют данные в восходящем порядке.

Сортировка простыми вставками

Из входного несортированного множества элементы берутся подряд и устанавливаются в массив так, чтобы не нарушать его упорядоченности.

Массив из ***n*** элементов делится на две части – сортированную и несортированную. На первой итерации алгоритма в сортированной части находится один первый элемент (массив из одного элемента всегда сортирован), остальные элементы находятся в несортированной части. Пусть ***i*** - индекс очередного элемента из несортированной части массива (элементы с индексами от ***i*** до ***n-1)***, которому надо найти место в сортированной части массива - индексы элементов от ***0*** до ***i-1***. Путем последовательно сравнения значения элемента с сортированными элементами, находится его место в сортированной последовательности, и элемент встраивается. Алгоритм повторяется, пока не будут встроены все элементы несортированной части массива.

Реализация алгоритма для обобщенного массива данных

```
template< class T >  //  T- обобщенный тип элементов массива
void insertSort(T* a, int size) {
 T tmp;
 for (int i = 1, j; i < size; ++i)  // i –индекс вставляемого элемента
 { tmp = a[i]; m=i;
   for (j = i - 1; j >= 0; --j) // поиск места элемента в готовой последовательности
     if( a[j] <= tmp ) break;     //место найдено
     else {a[m]=a[j];         //сдвиг больших элементов вправо
               m=j; }     //ищем место вставки  - индекс m
```

```
    a[m] = tmp; // место найдено, вставляем элемент
 } }
```

Среднее время сортировки пропорционально **$O(n^2)$,** дополнительная память при этом не используется. Хорошим показателем сортировки является весьма естественное поведение: почти отсортированный массив будет досортирован очень быстро. Это, вкупе с устойчивостью алгоритма, делает метод хорошим выбором при сортировке небольших массивов.

Из сложных методов, которые лучше подходят для сортировки больших массивов данных, рассмотрим метод ***быстрой сортировкой***.

Быстрая сортировка

Быстрая сортировка – это общее название ряда алгоритмов, которые отражают различные подходы к получению критичного параметра – опорного элемента, влияющего на производительность метода. Этот метод основывается на последовательном разделении сортируемого набора данных на блоки меньшего размера таким образом, что между значениями элементов разных блоков обеспечивается отношение упорядоченности.

Опорным элементом называется некоторый элемент массива, разделяющий массив на меньшие блоки. С точки зрения повышения *эффективности алгоритма* в качестве опорного элемента должна выбираться *медиана*, но без дополнительных сведений о сортируемых данных ее обычно невозможно получить. Поэтому можно выбирать, например, средний или последний по положению элемент или выбирать элемент со случайно выбранным индексом.

Алгоритм быстрой сортировки

Пусть дан массив ***a[n]*** размером ***n*** элементов.
Шаг 1. Выбирается опорный элемент массива.
Шаг 2. Массив разбивается на два – левый и правый – относительно опорного элемента. Реорганизуем массив таким образом, чтобы все элементы, меньшие опорного элемента, оказались слева от него, а все элементы, большие опорного – справа от него. А опорный элемент оказывается на своем месте в сортированном массиве.
Шаг 3. Далее повторяется шаг 1 для каждого из двух вновь образованных массивов. Каждый раз при повторении преобразования очередная часть массива разбивается на два меньших и т. д., пока не получится массив из двух элементов.

В среднем случае количество операций, выполняемых алгоритмом быстрой сортировки, определяется выражением $T(n) = O(1.4n \log n)$.

Быстрая сортировка, и все усовершенствованные на ее базе методы имеют один общий недостаток – невысокую скорость работы при малых значениях ***n***. Рекурсивная реализация быстрой сортировки позволяет устранить этот недостаток путем включения простого метода сортировки для частей массива с небольшим количеством элементов. Анализ вычислительной

сложности таких алгоритмов показывает, что если подмассив имеет десять или менее элементов, то целесообразно использовать простой метод, например, сортировку простыми вставками.

Выберем в качестве опорного - элемент, расположенный на средней позиции.

Реализация алгоритма для обобщенного массива данных

```
template<class T>       //  T- обобщенный тип элементов массива
void quickSortR(T* a, long N) {
// На входе - массив a[], a[N] - его последний элемент.
  long i = 0, j = N;
  T temp, p;
  p = a[ N>>1] ;    //определение  опорного элемента
   //процедура разделения
  do {    while ( a[i] < p ) i++;
          while ( a[j] > p ) j--;
          if (i <= j) {
         temp = a[i]; a[i] = a[j]; a[j] = temp;
         i++; j--;
            }
   } while ( i<=j );
  // рекурсивные вызовы, если есть, что сортировать
  if ( j > 0 ) quickSortR(a, j);
  if ( N > i ) quickSortR(a+i, N-i);
}
```

Метод неустойчив. Поведение довольно естественно, если учесть, что при частичной упорядоченности повышаются шансы «правильного» разделения массива на части. Сортировка использует дополнительную память, для реализации рекурсии.

Остальные методы рассмотрены в лекциях 26, 27.

Задача. В бинарном файле *binary.cpp* хранятся данные о студентах: номер зачетки, наименование группы, ФИО, размер стипендии. Вывести список данных о студентах на экран в порядке возрастания стипендии.

```
#include "stdafx.h"
#include <iostream>
#include <cstring>
#include <iomanip>
#include <fstream>
#include <stdlib.h
using namespace std;
struct stud {char fio[50], gr[8]; unsigned  int nz; float rs;} st;
int z = sizeof(stud);
char *sh[] = {
"                    СВЕДЕНИЯ О СТУДЕНТАХ                      ",
"╔═══════════╦═════════╦═════════════╦═════════════╗",
"║НОМЕР      ║ГРУППА   ║ФАМИЛИЯ      ║РАЗМЕР       ║",
"║ЗАЧЕТКИ    ║         ║ИНИЦИАЛЫ     ║СТИПЕНДИИ    ║",
"╠═══════════╬═════════╬═════════════╬═════════════╣",
"║           ║         ║             ║             ║",
"╚═══════════╩═════════╩═════════════╩═════════════╝"};
```

```
ifstream fin;                    // определение входного файлового потока
//---------------------------------------------------------------------------
void bubbleSort(stud* arr, int size) //метод сортировки пузырькового всплытия
{ stud tmp;     int pr;
for(int i = 0; i < size - 1; ++i) // i - номер прохода
 { pr=1;
  for(int j = size - 1; j >=i+1; --j) // внутренний цикл прохода
  if (arr[j - 1].rs > arr[j].rs) {  pr=0; tmp = arr[j - 1]; arr[j - 1] = arr[j]; arr[j] = tmp;}
  if(pr) break;
  } }
//------------------------------------------------------------------------------
// void psh( ) // вывод на экран строк шапки таблицы в файл
{ for(int i = 0;i < 5; i++) cout << sh[i] << endl;}
//--------------------------------------------------------------------------------------
// Сортировка и вывод записей файла
void sort_list( ) { int n = 0, i=0;              //счетчик структур в бинарном файле
cout << "\n             Список студентов :\n";
fin.open ("binary.cpp", ios::in|ios::binary); //открытие файла для чтения
if (!fin) {cout<<"Ошибка открытия бинарного файла для чтения"; exit(0);}
psh();                        // вывод строк шапки таблицы
while (!fin.eof( ))      //пока не достигли конца файла
{fin.read( (char*) &st, z);              //считываем структуру в переменную st
n++;}                                     //увеличиваем значение счетчика
stud*M=new stud[n]];                  //выделяем память на массив
fin.close();
fin.open ("binary.cpp", ios:: in|ios::binary);
while (!fin.eof( ))      {fin.read( (char*)(M+i), z);i++;}// заполнение массива
fin.close();
bubbleSort (M,n);// сортировка массива
psh();// вывод строк шапки таблицы
for (int i = 0; i < n; i++)   //выводим поля структуры в таблицу
cout << '|' << setiosflags(ios::left) << setw(13) << M[i].nz << '|'
 << setw(10) << M[i].gr << '|'  '<< setw(15) << M[i].fio << '|'  '
 << setw(14) << setprecision(2) << M[i].rs <<'|'  << endl;
cout << sh[5] << endl;
 delete [] M;}
int main ( ) {          //главная функция
sort_list( );
system("pause");
return 0;}
```

7.3. Задания для самостоятельного решения

1) Используя улучшенный вариант алгоритма быстрой сортировки, отсортировать массив структурных данных, считанных из файла. Данные включают характеристики студентов - ФИО, номер группы, оценки сессии, средний балл. Отсортировать массив по возрастанию среднего балла студентов.

2) Используя алгоритма сортировки методом Шелла, отсортировать динамический массив структурных данных, введенных с клавиатуры. Данные

включают характеристики библиотечных книг - название, автор, издательство, шифр. Отсортировать массив по шифру в алфавитном порядке.

3) Используя алгоритма сортировки методом пузырькового всплытия, отсортировать динамический массив структурных данных, считанных из файла. Данные включают характеристики товаров - Наименование, код, цена, производитель. Упорядочить массив по названию в алфавитном порядке.

4) Используя алгоритма сортировки методом слияния, отсортировать динамический массив структурных данных, считанных из файла. Данные включают характеристики автомобилей - марка, цвет, цена, производитель, мощность. Упорядочить массив по возрастанию мощности.

7.4. Контрольные вопросы

1) Что такое сортировка данных? Ключ сортировки.
2) Внешняя и внутренняя сортировки.
3) По каким критериям оцениваются алгоритмы сортировки?
4) Классификация алгоритмов сортировки.
5) Алгоритмы простым выбором, обменом, вставкой.
6) Алгоритм быстрой сортировки.
7) Сортировка слиянием.
8) Сортировка Шелла.

Литература

1. Надейкина Л.А. Программирование. Часть 2: учебное пособие – М.:МГТУ ГА, 2017, 84с.

2. Страуструп Б. Программирование: принципы и практика использования C++ 2-е издание, ISBN-13: 978-0321992789, Бином, Невский диалект, 2013,1312с.

3. Прата С. Язык программирования С. Лекции и упражнения. ISBN: 978-5-8459-1950-2 (рус.), Вильямс, 2015, 928 с.

4. Подбельский В.В. Стандартный Си++. М.: Финансы и статистика, 2008, 688с.

5. Городня Л.В. Парадигмы программирования. М.: НОУ "Интуит", 2016, 278 с.