МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ
ИМЕНИ В.Н. КАРАЗИНА

КРАТКИЙ СПРАВОЧНИК ПО ЯЗЫКУ ПРОГРАММИРОВАНИЯ **C++**

ХАРЬКОВ – 2003

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ
ИМЕНИ В.Н. КАРАЗИНА

КРАТКИЙ СПРАВОЧНИК ПО ЯЗЫКУ ПРОГРАММИРОВАНИЯ
C++

Методические указания для студентов I-IV курсов
специальности «Механика»

ХАРЬКОВ – 2003

УДК 004.43(03)
ББК 32.973-018.1.2
К 68

Короткий довідник з мови С++. Методичні вказівки для студентів I-IV курсів спеціальності “Механіка” / Укладач І.І. Ієвлев - Харків: ХНУ, 2003. - 50 с.

Данные методические указания имеют вид краткого справочника по языку программирования С++. Здесь приведены основные конструкции языка С++, сопровождаемые поясняющими примерами. Данные методические указания можно рассматривать как краткий конспект по элементам языка С++ для студентов, начинающих изучать программирование на СИ.

Рецензент: кандидат физ.мат.наук, доцент В. В. Камышан

Рекомендовано до друку кафедрою теоретичної механіки Харківського національного університету ім. В. Н. Каразіна (протокол №1 від 30.08. 2002)

Оглавление

Основные конструкции языка **5**

Константы *5*
Целые 5
Длинные целые 5
С плавающей точкой 5
Символьные 5
Строковые 5

Основные типы данных *5*
Основные типы и размеры данных 5
Указатели 5
Массивы 5
Массивы и указатели 6
Структуры 6
Поля бит структуры 6
Объединение 7
Перечисление 7
Переименование типов 7

Типы переменных *7*
Постоянные и временные переменные 7
Инициализация переменных 8
Внешние объекты 8

Операции *8*
Арифметические операции 8
Операция присваивания 9
Операции отношения 9
Логические операции 9
Побитовые операции 9
Адресные операции 9
Другие операции 10

Операторы *10*
Метка оператора 10
Составной оператор 10
Пустой оператор 10
break 10
continue 10
return 10
goto 10
Условный оператор if-else 11
switch 11
while 11
do-while 11
for 12

Функции *12*
Определение функции 12
Вызов функции 13
Функция main 13

Препроцессор *13*
Замена идентификаторов 13
Макросы 13
Включение файлов 13
Условная компиляция 14

Базовые классы *14*
Объявления класса 14
Данные-члены 15
Функции-члены 15
Конструкторы класса 16
Деструкторы 17
Дружественные функции 17

Производные классы *18*
Объявления 18
Функции-члены производного класса 18
Виртуальные функции и абстрактные классы 19
Конструкторы базового класса при наличии производных классов 19
Деструкторы базовых классов при наличии производных классов 20
Множественные базовые классы 20
Виртуальные базовые классы 20
Перегрузка функций 21

Библиотека ввода-вывода *23*
Библиотека C 23
Библиотека C++ 29

Структура программы на языке C++ **32**

Литература **34**

Основные конструкции языка

Комментарии

```
// текст          - до конца строки
/* текст */       - многостроковый
```

Константы

Целые

```
12    111   1070        десятичные
012   011   01070       восьмеричные
0x12  0x111 0x1070      шестнадцатеричные
```

Длинные целые

```
12L         5701L
```

С плавающей точкой

```
345.  -0.3142     2.9801e-7   -.32177
```

Символьные

```
'a'   'A'   '7'   'Б'   'ы'
Специальные:
 '\n' - перевод каретки
 '\t' - горизонтальная табуляция
 '\v' - вертикальная табуляция
 '\f' - перевод строки
 '\"' - кавычки
 '\0' - признак конца строки символов
 '\ddd' - ddd – любой набор 3-х восьмеричных цифр.
```

Строковые

```
"This is string"  "A"   "0123"
```

Основные типы данных

Основные типы и размеры данных

Тип данных	К-во бит	Диапазон
char	8	-127…127
unsigned char	8	0…256
int	16	±32,768
long	32	±2,147,483,648
foat	32	$\pm 3.4*10^{\pm 38}$
double	64	$\pm 1.7*10^{\pm 308}$

Указатели

<основной тип> *<идентификатор1>,*<идентификатор2>,…

Примеры:

указатель на переменную символьного типа

```
char *p;
```

указатель на указатель переменной символьного типа

```
char **t;
```

указатель на функцию

```
int (*f)();
```

Массивы

<основной тип>

<идентификатор1>[<целое1>],<идентификатор2>[<целое2>],…

Примеры:

одномерный массив из 50 целых
int a[50];

одномерный массив из 21 вещественных
double d[21];

одномерный массив из 20 символов
char s[20];

одномерный массив из 17 указателей на вещественную переменную
float *f[17];

двухмерный массив размера 2x3
int arr[2][3];

обращение к элементу массива
arr[0][2]=7;

Массивы и указатели

Примеры:

описание массива и указателя на целую
int A[10], *PA;

A это есть ***&A[0]*** (& - операция взятия адреса см. ниже), т.е. константа, равная адресу нулевого элемента массива.

Можно:	Нельзя:
PA=A	A++
PA=A+3	A=PA
PA++	
*(PA+i)	
PA=&A[3]	

Структуры

struct<имя структуры>{
<описание элементов структуры>
};

Пример:

```
struct book {

      char author[15];

      char *name;
      int  year;

};
```

описание переменной структурного типа
```
struct book univerlib;
struct book *town;
struct book regions[7];
```

обращение к элементам структурной переменной
```
univerlib.author="Иванов В.К.";
town->name="Вечный зов";
regions[0].year=1983;
```

Поля бит структуры

Пример:

```
struct bfield {
      unsigned int f1:10;
      unsigned int f2:6;
};
```

Здесь описаны 2 битовых поля для хранения целых переменных, при вычислениях они преобразуются в переменные типа unsigned int, а хранятся в битовых полях размера 10 и 6 бит.

Объединение

Описывают переменную, которая может иметь тип из некоторого множества типов.

union <имя объединения> {
<описание элементов объединения>
};

Примеры:

```
//определение типа
union bigname {
      long b1;
      char *b2:
};

//описание переменных типа объединение
union bigname x;
union bigname *y;
union bigname z[7];

//использование элементов переменных типа объединение
x.b1=123456;
y->b2="";
z[6].b1="AB";
```

Перечисление

Данные перечислительного типа могут принимать значения из некоторого множества значений.

enum <имя перечисления> {
<список значений>
};

Пример:

```
//описание типа
enum color {
      red, green, blue
};

//описание переменных
enum color picture, *page;

//использование переменных
picture=green;
*page=blue;
```

Переименование типов

typedef <старый тип> <новый тип>

Пример:

```
typedef long LARGE;
typedef char* string;
```

Типы переменных

Постоянные и временные переменные

Постоянные переменные хранятся в памяти до конца работы программы. К ним относятся переменные, в описании которых содержится ключевое слово static, или глобальные переменные, описанные вне блоков и имеющие область видимости, равную всему файлу.

К *временным переменным* относятся автоматические переменные (описанные внутри какого-либо блока), регистровые переменные и формальные параметры функций.

Примеры автоматических переменных:

```
{
int i; //область видимости – данный блок
}
register int k;
```

Замечание: к регистровым переменным нельзя применять операцию & (&k – ошибка!).

Инициализация переменных

Инициализацию можно применять ко всем переменным, кроме:

- формальных параметров функций;
- автоматических массивов, структур, объединений.

Примеры:

```
int i=1;
double pi=3.141592653;

//инициализация массива символов
char s[]="Привет!";

//неполная инициализация
char c[20]={'a','b','7'};

struct persons {
      int height;
      double weight;
};
//инициализация структурной переменной
struct persons man={175,75.3};
//инициализация массива структурных данных
struct persons fam[]= {{175,75.3},{170,72.3}};
```

Внешние объекты

extern <описание переменной>;

Объекты описаны в других модулях.

Примеры:

```
extern int i;
externchar *name;

extern int variable[];
//размеры массива можно не указывать

extern double func();
```

Операции

Арифметические операции

Обозначения:

e – любое выражение;
v – любое выражение типа lvalue (выражение, имеющее адрес);
i – целое число или символ;
a – арифметическое выражение;
p – указатель;
s – структура или объединение;
ps – указатель на структуру или объединение;
f – функция;
pf – указатель на функцию.

Операции:

ae1+ae2 – сложение;
ae1-ae2 – вычитание;

pe+ie - прибавление целого к адресу;
pe-ie - вычитание целого из адреса;
-ae - унарный минус;
ae1*ae2 – умножение;
ae1/ae2 – деление;
ai1%ai2 - остаток от деления (деление по модулю);
iv++ - эквивалентно iv=iv+1, значение равно iv;
pv++ - эквивалентно pv=pv+1, значение равно pv;
++iv - эквивалентно iv=iv+1, значение равно iv+1;
++pv - эквивалентно pv=pv+1, значение равно pv+1;
iv-- - эквивалентно iv=iv-1, значение равно iv;
pv-- - эквивалентно pv=pv-1, значение равно pv;
--iv - эквивалентно iv=iv-1, значение равно iv-1;
--pv - эквивалентно pv=pv-1, значение равно pv-1.

Операция присваивания

v=e

Операция	Эквивалентно
+=	av=av+ae (pv=pv+ie);
-=	av=av-ae (pv=pv-ie);
*=	av=av*ae;
/=	av=av/ae;
%=	iv=iv%ie;
>>=	iv=iv>>ie – сдвиг двоичного представления на ie бит вправо;
<<=	iv=iv<<ie – сдвиг двоичного представления на ie бит влево;
&=	iv=iv&ie – побитовое "И" двоичных iv и ie;
^=	iv=iv^ie – побитовое исключающее "ИЛИ" двоичных iv и ie;
|=	iv=iv|ie – побитовое "ИЛИ" двоичных iv и ie.

Операции отношения

Логическое выражение является ложным, если оно равно целому нулю. В противном случае оно является истинным. Константы false=0, true=1.

Операция	Использование
==	ie1==ie2
!=	ie1!=ie2
<	ae1<ae2
<=	ae1<=ae2
>	ae1>ae2
>=	ae1>=ae2

Логические операции

!ae - отрицание;
e1||e2 - "ИЛИ";
e1&&e2 - "И".

Пример:

```
if (p!=NULL && *p>7)  n++;
```

Побитовые операции

~ie - дополнение до единицы;
ie1 >> ie2 - сдвиг вправо на ie2 разрядов;
ie1 << ie2 - сдвиг влево на ie2 разрядов;
ie1 & ie2 - И;
ie1 | ie2 - ИЛИ;
ie1 ^ ie2 – исключающее ИЛИ.

Адресные операции

&v - адрес переменной v;
*pv - значение выражения, адресуемого указателем pv.

Другие операции

ae?e1:e2 - выражение равно e1, если ae истинно, и равно e2, если ae ложно.

e1,e2 - выполняется e1, затем e2. Выражение равно e2.

sizeof(e) - равно числу байт, необходимое для размещения данных типа переменной **e**.

sizeof(тип) - равно числу байт, необходимое для размещения объектов данного типа.

(тип)e - преобразование выражения e в указанный тип.

Операторы

Метка оператора

<идентификатор>: <оператор>;

Пример:

```
ABC2: x=1;
```

Составной оператор

{<оператор>;…;<оператор>;}

Пример:

```
{ x=1; y=x-1; z=sin(x); }
```

Оператор-выражение
оператор присваивания
оператор вызова функции

Пустой оператор

Это ;

break

Прекращает выполнение операторов switch, while, do, for.

Пример:

```
for (i=0;i<n;i++) {
      if (a[i]=b[i]==0) break;
}
```

continue

Передает управление на начало операторов цикла, вызывая начало новой итерации.

Пример:

```
for (i=0; i<n; i++) {
      if (a[i]!=0) continue;
      a[i]=b[i];
}
```

return

Прекращает выполнение функции и возвращает управление вызывавшей программе с возвращением выражения.

Пример:

```
return x+y*z;
```

goto

goto <метка>;

Оператор перехода на помеченный меткой оператор.

Пример:

```
goto ABC2;
```

Условный оператор ***if-else***

if (ae) <оператор> else <оператор>;
if (ae) <оператор>;

Примеры:

```
if (a[i]==x)
x=7;
else
x=a[i];

if (x>0) {
      if (y==1)
            z=5;
      else
            z=3;
}
else
      z=x;
```

switch

switch (<выражение>) {
case <константа>: <операторы>;
case <константа>: <операторы>;
** .**
** .**
default: <операторы>;
}

Сравнивает <выражение> с <константами> и передает управление на оператор, помеченный соответствующей <константой>.

Вариант ***default*** является необязательным и выполняется, если ни одна <константа> не равна <выражению>.

Примеры:

```
switch (x) {
      case 'A': c="Hello";       break;
      case "B": c="I am sorry!"; break;
      default:  c="Good buy!";
}
switch (x) {
      case 2:
      case 3: x=y;
}
```

while

while (<выражение>) <оператор>;

Если выражение истинно, то <оператор> выполняется до тех пор, пока <выражение> не станет ложным.

Примеры:

```
while (k<n) k++;
while (++i,i<n) ;
```

do-while

do <оператор> while (<выражение>);

Вначале выполняется <оператор>, затем проверяется <выражение>. Если <выражение> истинно, то <оператор> выполняется снова до тех пор, пока <выражение> не станет ложным.

Пример:

```
x=1;
do    {
      y *= 7;
      x += y*z;
} while (x<99);
```

for

for (<выражение1>;< выражение1>; <выражение1>) <оператор >;

<выражение1> - описывает инициализацию цикла,
<выражение2> - проверка условия завершения цикла,
<выражение3> - вычисляется после каждой итерации.

Данный оператор эквивалентен следующей последовательности операторов:

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

Примеры:

```
for (i=0; i<10; i++) printf("%d\n", x*x);
for (i=0; i<=20; a[i]=b[i], i++);
```

new
delete

Операции распределения памяти.

Операция	Содержание
new type	Выделяет место в памяти для типа type и возвращает адрес. При недостатке места в памяти возвращает нуль;
new type [ie]	Выделяет место в памяти для ie элементов типа type и возвращает адрес;
delete pv	Освобождает память, на которую указывает pv;
delete [] pv	Освобождает память от массива, на который указывает pv.

Пример:

```
void main ()
{
      char *ps;
      double **pv;

      //выделить память для строки
      ps = new char[37];
      strcpy(ps,"Копирование строки в память.");

      /* выделить память для двухмерного массива вещественных чисел */
      pv = new double* [3];
      for(int i=0;i<3;i++)
      if (!(pv[i] = new double [5])) {
            cout<<"Не хватает места в памяти!";
            exit(1);
      }

      //освобождение памяти
      delete ps;
      for(i=0;i<3;i++)
      delete pv[i];
      delete pv;
}
```

Функции

Определение функции

<тип> <имя функции>(<формальные параметры>) <тело функции>;

Примеры:

```
double distance(double x1, double y1, double x2, double y2)
{     return sqrt((x2-x1)*( x2-x1)+ (y2-y1)*( y2-y1));
}

void triangle(double a, double b, double c, double &p, double &sq)
{     p=(a+b+c)/2;
      sq=sqrt((p-a)*(p-b)*(p-c)*p);
}
```

Здесь &p, &sq – ссылки[1] на переменные вещественного типа. Функция не возвращает никакого значения (void), но изменяет значения переменных, на которые ссылаются параметры &p, &sq.

Вызов функции

<имя функции> (e1,e2,…)

(*<указатель на функцию>)(e1,e2,…)

Каждое выражение e1,…, представляющее фактические параметры функции, вычисляется и передается в стек. После выхода из функции эти значения в стеке теряются. Если тип функции не описан, то считается, что он равен int.

Функция main

Каждая программа начинает работу с функции main(). Функция может иметь параметры argc, argv,envp. Эти параметры имеют тип

```
int argc;    //число параметров
char **argv; //вектор параметров-строк
char **envp; //вектор переменных среды.
```

Препроцессор

Если первым символом строки является знак #, то эта строка является командой препроцессора (т.е. выполнение команды происходит на стадии компиляции программы).

Замена идентификаторов

#define <идентификатор> <строка>

Примеры:

1) *#define ABC 100*

Заменяет каждое вхождение идентификатора ABC в программе на 100.

2) *#undef ABC*

Отменяет предыдущее определение для идентификатора ABC.

3) *#define DEBUG*

Макросы

#define <идентификатор1> (<идентификатор2, идентификатор3,…>) <строка>

Примеры:

1) *#define abs(A) ((A)>0?(A):(A))*

каждое вхождение в программе abs(arg) заменяется на выражение

((arg)>0?(arg):(arg))

2) *#define MAX(A,B) ((A)>(B)?(A):(B))*

Включение файлов

#include <имя файла>

Примеры:

1) *#include <math.h>*

Препроцессор в данном месте программы заменяет эту строку содержимым файла math.h. Угловые скобки говорят о том, что данный файл будет разыскиваться в некотором стандартном каталоге.

[1] Ссылка – это другое имя переменной, на которую она ссылается. Обязательно до использования ссылки она должна быть инициализирована какой либо переменной:
int i, &i2=i; i2=3; // будет i=3 и i2=3

2) *#include "mylib.h"*

Здесь также в программе заменяется данная строка на содержимое файла mylib.h. Однако файл разыскивается в том же каталоге, где находится основная программа. Если там этого файла нет, то поиск производится вначале в каталоге, указанном в опциях компилятора, а затем в стандартном каталоге.

Условная компиляция

#if <константное выражение>

Пример:

#if ABC+3

истина, если выражение ABC+3 не равно нулю.

#ifdef <идентификатор>

Выражение истинно, если идентификатор определен ранее командой #define.

Пример:

#ifdef ABC

#ifndef <идентификатор>

Выражение истинно, если к данному моменту идентификатор не определен.

Пример:

```
#ifndef ABC
#else
.
.
#endif
```

Если предшествующие операторы #if, #ifdef, #ifndef дают истинное значение, то строки между #else … #endif игнорируются. В противном случае пропускаются строки до оператора #else.

Пример:

```
#ifdef DEBUG
  #define DEBUGGING
#else
  #define WORKING
#endif
```

Базовые классы

Объявления класса

Класс – это новый тип, определяемый пользователем и содержащий набор данных (данные-члены) и функций (функции-члены), обрабатывающих эти данные.

Для членов класса устанавливаются ограничения доступа к ним:

private – члены класса доступны только членам класса и дружественным функциям;
protected - члены класса доступны только членам класса и членам производного класса;
public - члены класса доступны всем функциям;
published - [2].

Примеры:

1) объявление класса

```
class Phone {
//private - по умолчанию
      int area, exchange, line;
protected:
      enum HookState {onHok, offHook}
      handset;
public:
```

[2] См. Основы программирования в CBuilder'е.

```
      //конструктор класса
      Phone (int a, int e,int l) {
            area=a, exchange=e, line=l;
            handset=onHook;
      }
      //функции-члены класса
      int OffHook() {
      return handset==offHok; }

      void GiveDailTone();
      long AcceptDigits();
      void Ring();
};
```

2) объявление переменной класса Phone

```
Phone home(516,555,8858);
```

Данные-члены

- Могут находиться в private, protected, public частях класса.

Пример:

```
home.area=77; //ошибка: private
home.handset=ofHook; //ошибка: protected
if (home.OffHook())  //правильно
  home.GiveDialTone();
```

- Могут иметь статический класс памяти (static). Статические члены класса используются совместно всеми объектами данного класса!
  К ним возможен доступ через имя класса (с использованием операции разрешения доступа "::"), а не через объекты класса;
- Могут быть объявлены как const. Константные члены-данные должны быть инициализированы при каждом определении конструктора класса.

Пример:

```
class Phone {
  const int area, exchange, line;
  //…
public:
  Phone(int a, int e, int l) ://далее инициализация
    area(a), exchange(e), line(l) {
   // тело конструктора
   }
// …
};
```

Функции-члены

- Имеют доступ ко всем данным-членам класса.
- Могут относится к private, protected, public частям класса.
- Могут быть определены как внутри класса, так и вне в отдельном файле.
- Имеют неявно объявленную переменную this – указатель на объект класса, членом которого является данная функция.
- Могут быть объявлены static. Только такие функции могут непосредственно обращаться к статическим данные класса. Статические функции-члены не могут быть объявлены как виртуальные (virtual). К таким функциям можно обращаться непосредственно через имя класса, а не имя объекта класса.
- Могут быть объявлены как виртуальные[3].

Пример:

1)

```
class Person {
protected:
  int BirthDay, BirthMonth, BirthYear;
  float Weight;
  int Height;
```

[3] См. производные классы.

```
public:
  Person(int aBirthYear) { BirthYear=a BirthYear; }
  int setBirthYear (int aBirthYear) {
      BirthYear= aBirthYear;
return BirthYear;
  }
  int getBirthYear () { return BirthYear; }
  virtual int getBirthMonth();
};

class DetailPerson : public Person {
  char *name;
public:
  DetailPerson(int aBirthYear, int aBirthMonth,
               int aBirthDay) {
  {   Person::Person(aBirthYear);
      BirthMonth=aBirthMonth;
      BirthDay=aBirthDay;
  }
  void SetName(char *aName) { name=aName; }
};

//объявление объектов класса DetaiPerson
DetaiPerson Anna(1986,11,15), Sveta(1966,1,12);

//использование функций класса
Anna.SetName("Петрова Анна Васильевна");
```

Конструкторы класса

- Инициализируют объект класса с необязательным списком аргументов. Аргументы могут иметь значение по умолчанию.
- Автоматически вызываются при объявлении объекта класса.
- Имеют то же имя, что и класс, для которого они определены.
- Могут относиться к private, protected, public частям класса.
- Могут быть определены как внутри, так и вне объявления класса.
- Не может быть объявлен ни static, ни virtual, ни const.
- Константные данные класса должны быть инициализированы в каждом определении конструктора. список таких данных отделяется от списка аргументов конструктора двоеточием, разделяются они запятыми и после каждого имени данных в круглых скобках ставится значение.

Пример:

```
class IntStack {
  const int size;
  int *v;
  // …
  IntStack(int aSize) : size(aSize) {
     v=new int[size];
};
```

- Конструкторы, не имеющие аргументов или все аргументы имеют значение по умолчанию, называют *конструкторами по умолчанию*. Конструкторы по умолчанию нужны, когда создается массив объектов данного типа.
- *Конструктором копирования* называют конструкторы, среди аргументов которых имеется хотя бы одна ссылка на объект данного класса, а остальные аргументы имеют значения по умолчанию. Конструктор копирования служит для создания одного объекта из другого объекта того же класса (копирование объектов). Конструктор копирования должен быть определен, если класс содержит данные-члены, являющиеся указателями на динамически распределяемую память.

Пример:

```
class IntArr {
  const int size;
  int *v;
public:
  //конструктор по умолчанию
  IntArr(int r=10){ size=r; }

  //конструктор копирования
  IntArr(IntArr &iA, int r=10){ v=iA.v; size=r; }
};

//определение объектов класса
IntArr m(100);//число элементов массива
              //больше 10
IntArr c(m);// объявить c=m или
IntArr c=m;
```

Деструкторы

- Выполняют действия перед разрушением объекта.
- Автоматически вызываются компилятором:
  - при выходе из области видимости объекта;
  - при создании временных объектов при передаче аргументов функциям;
  - при выполнении операции delete для объектов, размещенных в динамической памяти.
- Имеют то же имя, что и класс, дополненное впереди знаком "~".
- Не имеют возвращаемого значения.
- Не могут быть виртуальными.

Пример:

```
class IntStack {
  int *v, size, top;
public:
  IntStack(int aSize=20);
  ~IntStack() { delete[] v;}
};
//использование
void main() {
  IntStack s(100), *ps=new IntStack(50);
  //…
  delete ps; //вызывается деструктор для ps
}  //неявно вызывается деструктор для s.
```

Дружественные функции

- Имеют доступ ко всем членам класса, хотя сами не являются функциями-членами. Содержат ключевое слово friend.
- Могут быть дружественными к более, чем одному классу.
- Объявляются внутри класса, к которому они имеют доступ.
- Не наследуются производными классами.

Пример:

```
class Person {
  int secret:
public:
  friend void Func(Person&);//объявление функции
};
//описание функции
void Func(Person &p) {
  ++p.secret;
}
```

Производные классы

Объявления

Предоставляют возможность создания вариации некоторого базового класса.
Вводят новый тип.
Устанавливают ограничения доступа:

- **private** – открытые (public) и защищенные (protected) члены базового класса становятся закрытыми (**private**) в производном классе;
- **protected** - открытые (public) и защищенные (protected) члены базового класса становятся защищенными (**protected**) в производном классе;
- **public** - открытые (public), защищенные (protected) и закрытые (private) члены базового класса становятся открытыми, (**public**), защищенными (**protected**) и закрытыми (**private**) в производном классе.

Указанную зависимость можно отразить следующей таблицей

Базовый класс	Доступ к членам в базовом классе	Доступ к членам в производном классе
private	private	private
	protected	private
	public	private
protected	private	private
	protected	protected
	public	protected
public	private	private
	protected	protected
	public	public

Объявление производного класса отличается от объявления базового класса тем, что после имени производного класса ставится двоеточие, а затем перечисляются имена базовых классов, разделенные запятыми, с указанием ограничения доступа.

Пример:

```
//базовый класс
class Phone {
// …
};

//производный класс
class PayPhone:public Phone {
// …
};
```

Функции-члены производного класса

- Они могут перекрывать одноименные функции базового класса.
- Могут вызывать функции-члены базового класса, используя операцию разрешения доступа ("::").

Пример:

```
//базовый класс
class Library {
  char author[15];
public:
  char* GetName() { return author; }
};

//производный класс
class MyLibrary : public Library {
  char Name[30];
  char FullName[100];
public:
  MyLibrary(char *aAuthor, char *aName)
  : Name(*aName)
  { Library::author=aAuthor; }
```

```
  char *GetName() {
    FullName=Library::GetName()+
             ". "+Name;
    return FullName;
  }
};

//описание объекта и его использование
MyLibrary mbook("Пушкин А.С.",
"Избранное");
//дальше присвоение названия книги
//переменной nameBook
char *nameBook=*mbook.GetName();
```

Виртуальные функции и абстрактные классы

Виртуальной является функция-член класса, определение которой производится в производных классах. Объявление виртуальной функции содержит ключевое слово virtual, а в объявлении виртуальной функции необходимо приравнять ее нулю.

Абстрактным называют класс, среди функций-членов которого имеется хотя бы одна виртуальная функция.

Пример:

```
//абстрактный класс
class Shape {
protected:
  struct Point {
    int x,y;
  } center;
public:
  //конструктор класса
  Shape(Point c={1,1}) : center(c) {}
  //виртуальная функция
  virtual Shape &Draw()=0;
};

//производный класс
class Circle : public Shape {
  int radius;
public:
  Circle(Point c, int r=1);
  Shape& Draw(); //здесь функция уже должна
                 //определяться
};
//использование классов
Shape a; //ошибка: абстрактный класс
struct {int x,y;} cen={12,17};
Circle c(cen,3);
Shape *pc=&c; //допустимо
```

Конструкторы базового класса при наличии производных классов

- Вызываются перед вызовом конструктора производного класса.
- Должны быть явно указаны в каждом определении конструктора производного класса, если у базового класса нет конструктора по умолчанию. Имя конструктора базового класса отделяется от имени конструктора производного класса двоеточием, а его аргументы перечисляются в скобках. Конструкторы множественных базовых классов отделяются друг от друга двоеточиями.

Пример:

```
class Phone{
//..
public:
      Phone(int, int, int); //Имеет аргументы
//..
```

```
};

//Производный класс
class PayPhone {
//..
public:
      PayPhone(int a, int b, int c):Phone(a,b,c);
//..
};
```

Деструкторы базовых классов при наличии производных классов

- Вызываются после деструкторов производных классов.
- Деструкторы абстрактных классов всегда должны быть виртуальными.

Множественные базовые классы

Производный класс может наследовать несколько базовых классов. Имена базовых классов в определении производного класса перечисляются после двоеточия через запятую.

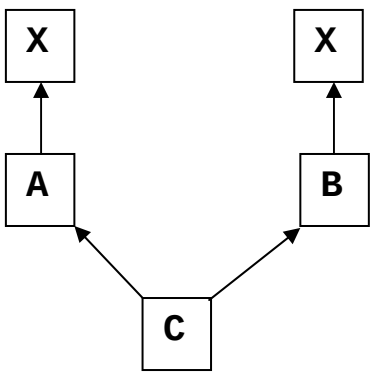
Пример:

```
class A {
public:
      void f();
};
class B {
public:
      void f();
};

class C:public A, public B {
//..
};

//Объявление объекта класса C
C cc;
cc.f();     //неправильно
cc.A::f();  //правильно
cc.B::f();
```

- Для классов, порожденных от производных классов, имеющих общий базовый класс, существует два экземпляра этого базового класса.

Пример:

```
class X {
//..
};
class A : public X {
//..
};
class B : public X {
//..
};
class C : public A, public B {
//..
};
```



Графическая зависимость производного класса C от базового X

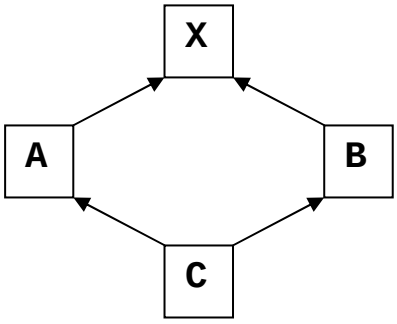
Виртуальные базовые классы

Для классов, порожденных от производных классов, имеющих общий виртуальный базовый класс, существует только один базовый класс. Виртуальный базовый

класс объявляется словом virtual перед спецификатором ограничения доступа (private, protected, public) в производных классах.

Пример:

```
class X {
//..
};
class A : virtual public X {
//..
};
class B : virtual public X {
//..
};
class C : public A, public B {
//..
};
```

Графическое изображение зависимости класса C от виртуального базового класса X

Позволяет расширить операции на классы. Осуществляется путем объявления функции, состоящей из ключевого слова operator, за которым стоит одна из встроенных операций (+,-,*,/,[],?:,::,sizeof).

Унарные операции

- Могут быть объявлены как нестатические функции члены, не имеющие аргумента, т.е. @x воспринимается как x.operator@() для любой операции @.
- Могут быть объявлены как функции, не являющиеся членами класса и имеющими один аргумент. Этот аргумент является либо переменной какого либо класса, либо ссылкой на него. Операция @x воспринимается как operator@(x) для любой операции @.

Примеры:

```
class X {
      X operator-(); // унарный минус
      X operator&(); //вычисление адреса
};
class y {
      //унарный минус
      friend Y operator-(Y&);

      //вычисление адреса
      friend Y operator&(Y&);
};
```

Бинарные операции

- Могут быть объявлены как нестатические функции-члены, имеющие один аргумент, т.е. x@y воспринимается как x.operator@(y) для любой операции @.

- Могут быть объявлены как функции, не являющиеся членами класса и имеющими два аргумента. Эти аргументы должны быть или переменными типа класса или ссылками на них. Для любой операции @ выражене x@y воспринимается как operator@(x,y).

Пример:

```
class X {
      //бинарный минус
      X operator-(X&, X&);
};

class Y {
      //бинарный минус
      friend Y operator-(Y&, Y&);
};
```

Операция присваивания

- Используется для присвоения одного объекта класса другому.
- Если не определена пользователем, то выполняется присваивание значений данных-членов одного объекта данным другого.
- Должна быть определена пользователем, если класс содержит указатели на динамически распределяемую память.
- Не наследуется в производных классах.

Пример:

```
class intStack {
      int *v, size, top;
public:
//..
      intStack operator=(const intStack&);
};

//Определение операции
intStack intStack::operator=(const
      intStack &from) {
      //from не равно from
      if (this != &from){
            delete[] v;
            v=new int[size=from.size];
            for (register int i=0; i<size;
            i++)
            v[i]=from.v[i];
            top=from.top;
      }
      return *this;
      /*Позволяет использовать множественное присваивание*/
}
//Объявление объектов и применение операции =
intStack a(10),b,c;
c=b=a;
```

Операция индексации

Должна быть определена как нестатическая функция-член класса. Если она возвращает ссылку на класс, то операция индексации может использоваться в операции "=" с обеих ее сторон.

Пример:

```
class String {
      char *s;
public:
      String(char);
      String(const char*);
      char& operartor[](int pos){ return s[pos];}
};

String show("figure");
show[0]='F';
```

Библиотека ввода-вывода

Библиотека С

При использовании функций библиотеки ввода-вывода **С** модуль программы должен включать в себя файл stdio.h с помощью команды препроцессора

#include <stdio.h>

Доступ к файлам

fopen – открыть файл filename и ассоциирует с ним поток ввода-вывода:

FILE *fopen(const char *filename, const char *mode);

Строка mode может иметь одно из следующих значений:

Значение	*Описание*
r	Открыть файл только для чтения.
w	Создать файл для записи. Если файл с ука занным именем уже существует, то он будет перезаписан.
a	Открывает файл для дозаписи в конец файла. Если файл не существует, то он будет создан.
r+	Открывает существующий файл для чтения-записи.
w+	Открывает файл для чтения-записи. Если файл с указанным именем существует, то он будет перзаписан, если не существует, то будет создан.
a+	Открывает файл для добавления записи в конец файла. Если файл с указанным именем не существует, то он будет создан.

Для того, чтобы файл был записан в текстовом формате, добавьте к mode букву t (rt w+t и т.д.). Соответственно, для задания бинарного формата добавьте к mode букву b (wb a+b и т.д.).

Если t или b не указаны в mode, то режим вывода определяется значением глобальной системной переменной _fmode. Если _fmode имеет значение O_BIINARY, файлы открываются как бинарные. Если _fmode имеет значение O_TEXT, то файлы представляются как текстовые. Константы O_... определены в файле fcntl.h.

Возвращаемое значение:

- если нет ошибок, то указатель на вновь открытый поток;
- NULL – если имели место ошибки операции.

freopen – закрыть поток stream и открыть новый файл newfile того же типа type:

FILE *freopen(char *newfile, int type, FILE *stream);

fclose – закрыть поток stream:

FILE *fclose (FILE *stream);

feof – проверка достижения конца файла. Возвращает true (не 0)при достижении конца файла, false (0)- в противном случае.

int feof(FILE *stream);

Пример:

```
#include <string.h>
#include <stdio.h>

int main(void)
{
   FILE *fp;
   char buf[11] = "0123456789";
```

```
   /* создание файла для записи 10 байт */
   fp = fopen("DUMMY.FIL", "w");
   fwrite(&buf, strlen(buf), 1, fp);

   /* закрытие файла */
   fclose(fp);
   return 0;
}
```

size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream); - дописывает в файл stream n раз объект размера size байт (всего n x size байт), на который указывает указатель ptr.

Возвращаемое значение - число объектов, записанных в файл.

int fprintf(FILE *stream, const char *format[, argument, ...]); - записывает ряд аргументов каждый в своем формате (см. ниже) в файл (поток stream).

Возвращаемое значение:

- Число выведенных в файл байт.
- EOF – в случае ошибки.

Пример:

```
#include <stdio.h>

struct mystruct
{
  int i;
  char ch;
};

int main(void)
{
   FILE *stream;
   struct mystruct s;

   /* открывает файл TEST.$$$ */
   if ((stream = fopen("TEST.$$$", "wb")) == NULL) {
      fprintf(stderr, "Не могу открыть файл.\n");
      return 1;
   }
   s.i = 0;
   s.ch = 'A';

   /* запись структуры в файл */
   fwrite(&s, sizeof(s), 1, stream);

   /* закрыть файл*/
   fclose(stream);
   return 0;
}
```

int fscanf(FILE *stream, const char *format[, address, ...]); - считывает ряд входных данных из файла (поток stream) в соответствии со спецификациями format (см. scanf) и записывает их по указанным адресам.

Возвращаемое значение:

- Число успешно введенных данных.
- EOF – если читается символ конца файла.
- 0 - если не найдено в файле ни одного поля данных.

Пример:

```
#include <stdlib.h>
```

```
#include <stdio.h>

int main(void)
{
   int i;

   printf("Введите целое число: ");

   /* чтение целого числа с клавиатуры */
   if (fscanf(stdin, "%d", &i))
      printf("Было введено целое число: %i\n", i);
   else
   {
      fprintf(stderr,
      "Ошибка чтения числа с клавиатуры.\n");
      exit(1);
   }
   return 0;
}
```

size_t fread(void *ptr, size_t size, size_t n, FILE *stream); - читает n блоков, каждый размера size байт, из потока stream в блок памяти, на который указывает указатель ptr.

Возвращаемое значение – число прочитанных блоков (не байт).

Пример:

```
#include <string.h>
#include <stdio.h>

int main(void)
{
   FILE *stream;
   char msg[] = "test";
   char buf[20];

   if ((stream = fopen("DUMMY.FIL", "w+"))
       == NULL) {
      fprintf(stderr, "Не могу открыть файл.\n");
      return 1;
   }

   /* запись данных в файл */
   fwrite(msg, strlen(msg)+1, 1, stream);

   /* поиск начала файла */
   fseek(stream, SEEK_SET, 0);

   /* чтение данных из файла и отражение их */

   fread(buf, strlen(msg)+1, 1, stream);
   printf("%s\n", buf);

   fclose(stream);
   return 0;
}
```

fgets –прочитать строку из потока, включая символ новой строки. Чтение прекращается, когда будет прочитано n-1 символлов или встретится символ новой строки:

```
char *fgets(char *s, int n, FILE *stream);
```

Пример:

```
#include <string.h>
#include <stdio.h>
```

```
int main(void)
{
   FILE *stream;
   char string[] = "test";
   char msg[20];

   /* открытие файла для дозаписи */
   stream = fopen("DUMMY.FIL", "w+");

   /* запись строки в файл */
   fwrite(string, strlen(string), 1, stream);

   /* возврат на начало файла */
   fseek(stream, 0, SEEK_SET);

   /* чтение строки из файла */
   fgets(msg, strlen(string)+1, stream);

   /* вывод на экран */

   printf("%s", msg);

   fclose(stream);
   return 0;
}
```

gets – прочитать строку из стандартного файла ввода stdin:

```
char *gets(char *s);
```

fputs – записать строку в поток stream:

```
int fputs (char *s, FILE *stream);
```

puts – записать строку в стандартный поток вывода stdout:

```
int puts (char *s);
```

fgetc – прочитать символ из потока:

```
int fgetc (FILE *stream);
```

getc – прочитать символ из стандартного потока ввода:

```
int getc ();
```

getchar – прочитать символ из стандартного потока:

```
int getchar();
```

fputc – записать символ в поток stream:

```
int fputc(int c, FILE *stream);
```

putc – записать символ в стандартный поток:

```
int putc(char c);
```

Пример:
создание дубля файла

```
#include <stdio.h>

int main(void)
{
```

```
   FILE *in, *out;
   if ((in = fopen("TESTFILE.DAT", "rt"))
       == NULL) {
      fprintf(stderr, "Не могу открыть файл.\n");
      return 1;
   }

   if ((out = fopen("TESTFILE.BAK", "wt"))
       == NULL) {
      fprintf(stderr, " Не могу открыть файл.\n");
      return 1;
   }

   while (!feof(in))
      fputc(fgetc(in), out);

   fclose(in);
   fclose(out);

   return 0;
}
```

putchar – записать символ в стандартный поток:

int putchar(int c);

Форматированный вывод

Запись в стандартный поток вывода stdout:

int printf(char *format, arg,…);

Запись в поток stream:

int fprintf(FILE *stream, char *format, arg,…);

Запись аргументов в строку символов:

int sprintf(char *s, char *format, arg,…);

Аргумент format (спецификатор)

%[выравнивание] [ширина] [доп.признаки] [символ преобразования]

где
выравнивание – вправо по умолчанию, символ "-" – влево;
ширина – целое, равное минимальному числу выводимых символов (если ширина выводимой переменной больше *ширины*, то выводится необходимое для отражения переменной число символов);
доп.признаки, символ преобразования - (см.примеры).

Примеры:

Спецификатор		Результат
Символы	%c	A
	%3c	__A
	%-3c	A__
Строки	%s	a1a2a3a4a5a6a7bcd
	%10s	a1a2a3a4a5a6a7bcd
	%-10.5s	abcde_____
	%10.5s	_____abcde
Целые	%d	1234567
	%3d	_12
	%-3d	12_
	%+3d	+12

Целое без знака — *%[-][#][ширина][l]{u|o|x|X}*[4]
- определяет вывод начального нуля для восьмеричных или шестнадцатеричных чисел,
u – десятичное без знака,
o – восьмеричное без знака,
x – шестнадцатеричное без знака,
X – шестнадцатеричное без знака с прописными буквами A-F

Спецификатор	Результат
%u	1234567890
%o	123456701
%#o	0123456701
%x	2e35f90
%#X	02E35F90

С плавающей запятой

%[-][+][#][ширина][.точность]{f|e|E|g|G}

- будут выводиться завершающие нули дробной части и всегда десятичная точка,
+ - для положительных чисел будет выводится знак +, для отрицательных всегда выводится знак -,
f – число с фиксированной точкой [-]ddd.ddd,
e – экспоненциальный формат [-]d.dddde{+|-}dd,
E – [-]d.ddddE{+|-}dd,
g – наиболее короткий формат из f или e,
G – наиболее короткий формат из f или E.

Спецификатор	Результат
%f	1234.5435
%.1f	1234.5
%E	1.23456E-03
%.3e	1.234e+03

Форматированный ввод

Ввод данных из стандартного потока stdin:

scanf(char *format,pointer,…);

Ввод данных из потока stream:

fscanf(FILE *stream,char *format,pointer,…);

Ввод данных из строки s:

sscanf(char *s,char *format,pointer,…);

Для использования этих функций требуется включение файла stdio.h:

#include <stdio.h>

Число и типы аргументов функций должны соответствовать спецификациям в форматной строке format, как и в случае функций вывода данных.

Пример:

Входной поток содержит данные
12.45 1048.73 AE405271 438

Вызов функции
*scanf("%f%*f%%s%d",&x,&id,&n);*
Переменным присваиваются значения
x = 12.45
Пропускается *1048.73* [5]
id = AE405271

[4] []-означает необязательный элемент, | - один из элементов.
[5] символ * в спецификации означает пропуск поля при вводе.

n = 438

Библиотека C++

В C++ ввод-вывод осуществляется через библиотеку классов:

ios	базовый потоковый класс
istream	класс потока ввода
ostream	класс потока вывода
iostream	класс потока ввода и вывода
ifstream	класс файла ввода
ofstream	класс файла вывода
fstream	класс файла ввода и вывода

Общие функции классов ввода-вывода

int eof()

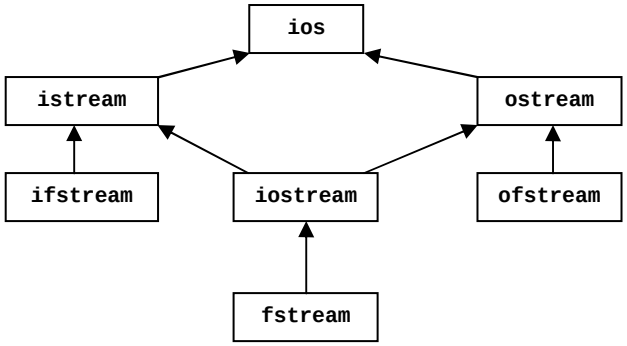
Возвращает ненулевое значение, если в потоке обнаружен символ конца файла.

int fail()

Возвращает ненулевое значение, если при выполнении операции с потоком произошла ошибка.

int good()

Возвращает ненулевое значение, если при операциях с потоком не было ошибок.



Иерархия классов ввода-вывода

Потоковый вывод[6]

Может применяться с одним из предопределенных выходных потоков из класса ostream или iostream:
cout – стандартное устройство вывода;
cerr – стандартное устройство сообщения ошибок.

Операция вставки "<<" записывает последовательность символов в указанный поток.

- Это перегруженная операция <<.
- Предопределена для всех простых типов и указателей на переменные типа char.
- Может быть перегружена для классов.

Примеры:

```
1) cout<<"Hello!\n";
   cerr<<"Плохой оператор:"<<op<<'\n';

2) class Phone {
      int area, exchange;
      //..
      friend ostream& operator<<(ostream&,
            const Phone&);
   };

  ostream& operator<<(ostream& os,
        const   Phone &p) {
        os<<"("<<p.area<<<<")"<<p.exchange<<"\n";
        return os;
  }
```

[6] Здесь отражены наиболее часто встречаемые функции потоков ввода-вывода. Подробнее смотрите подсказку соответствующего программного обеспечения.

```
  Phone home;
  cout<<"Home number:"<<home<<"\n";
```

ostrem& put(char) – вывод символа в поток.

Примеры:

```
cout.put('\n');
cout.put('A');
```

Потоковый ввод

Использует классы istream или iostream. Может применяться с предопределенным входным потоком cin или входным потоком, определенным пользователем.

Операция извлечения ">>" читает последовательность символов из потока.

- Это перегруженная операция >>.
- Она предопределена для всех простых типов и указателей на переменную типа char.
- Имеет ненулевое значение до тех пор, пока не встретит символ конца файла.
- Для классов может быть перегружена.
- Устанавливает поле failbit класса в соответствующее значение, если считываемый символ имеет не соответствующий тип.

Пример:

```
char c;
while (cin>>c) cout<<c;
```

int get() – функция, которая извлекает символ из входного потока (в том числе и пробел, табуляцию,…). Возвращает значение EOF, если достигнут конец файла.

Пример:

```
int c;
while (c=cin.get() != EOF) cout.put(c);
```

int gcount()

Возвращает число символов, прочитанных из потока во время последней операции неформатированного ввода.

Файловый ввод-вывод

Осуществляется посредством функций класса fstream[7] (**ofstream** – для вывода, **ifstream** – для ввода).

Примеры:

1)

```
#include<iostream>
#include<fstream>
void main ( )
{
    // создание потока ввода-вывода
    fstream inout("fstream.out");
    // вывод символов в поток
    inout << "Объектно-ориентированный язык." << endl;
    inout << "Визуальное программирование." << endl;
    inout << "История создания." << endl;

    char p[100];
    // возврат на начало файла
    inout.seekg(0);

    // чтение первой строки
    inout.getline(p,100);

    // вывод первой строки на экран
    cout << endl << "Привет!" << endl;
    cout << p;
    fstream::pos_type pos = inout.tellg();
```

[7] Подробнее см. подсказку программного обеспечения.

```
    // чтение второй строки
    inout.getline(p,100);

    // вывод первой строки на экран
    cout << endl << "Еще привет!" << endl;
    cout << p;

    // чтение третьей строки
    inout.getline(p,100);

    // вывод первой строки на экран
    cout << endl << "Трижды привет!" << endl;
    cout << p;

    /* установить указатель потока на начало второй
       строки */
    inout.seekp(pos);
    // заменить вторую строку
    inout << "Нет более печальнее на свете…" << endl;
    // заменить третью строку
    inout << "…чем повесть о Ромео и Джульете.";

    // переход на начало файла
    inout.seekg(0);
    // вывод всего содержанияфайла на экран
    cout << endl << endl << inout.rdbuf();
}
```

```
2)
//попытка открыть файл для записи
ofstream inFile("MyFile");
if (!inFile) {
   cerr<<"Не могу открыть файл
   /"MyFile/".\n";
   exit(1);
}
  inFile<<"Hello!\n";

3)
//попытка открыть файл для чтения
ifsream fromFile("MyFile");
if (!fromFile) {
    cerr<<"Не могу открыть файл
    /"MyFile/".\n";
    exit(1);
}
int a,b;
fromFile>>a>>b;
```

Форматирование ввода-вывода

Форматирование устанавливается при помощи функций-членов flags(), setf(), unsetf().

Функция setf() устанавливает следующие флаги, которые являются данными-членами класса *ios*:

skipws пустые промежутки при вводе будут пропускаться;

dec выбор системы счисления (десятиная,
oct восьмеричная,
hex шестнадцатеричная);

showbase восьмеричные числа начинаются с 0, шестнадца
теричные с 0x или 0X;

showpoint всегда выводится десятичная точка;

showpos для положительных чисел всегда выводится знак "+";

scientific экспоненциальный формат представления чисел с плавающей запятой;

fixed после десятичной точки число цифр устанав ливается методом precisiob().

long flags() – возвращает текущие флаги форматирования.
long flags(long)
Устанавливает флаги форматирования.

long setf(long)
Устанавливает флаги форматирования и возвращает их старое значение.

long unsetf(long)
Выключает указанные флаги форматирования и возвращает их предыдущее значение.

int precision(int)
Устанавливает число значащих цифр при выводе чисел с плавающей запятой и возвращает его старое значение.

int precision()
Возвращает текущее число значащих цифр.

Примеры:

```
1)   long oldFlags=cout.flags();
     //.. изменяются флаги
     cout.flags(oldFlags);
2)   long oldFlags=
     cout.setf(ios::showbase);
3)   cout.unsetf(ios::showbase &
     ios::uppercase);
```

Структура программы на языке C++[8]

Простейшая программа может содержаться в одном файле. Программа представляет собой совокупность директив препроцессора, описаний и функций, среди которых обязательно должна присутствовать функция main. Выполнение программы начинается с выполнения этой функции.

Пример:

Пусть программа содержится в файле hello.cpp

```
//директивы препроцессору
#include …
#define …
//глобальные переменные
const pi = 3.141592653;
int N;
…
//прототипы функций[9]
//главная функция
void main()
{
…
}
//описания других функций
```

[8] Детали создания выполняемых программ отличаются для различных программных пакетов и их версий (см. подсказку используемого программного обеспечения).
[9] Прототип функции фактически совпадает с заголовком функции.

```
double func1(){
…
}
…
```

Рекомендуется описания типов, классов, прототипы функций, макроопределения выносить в отдельный включаемый файл (*.h). Сложная программа может быть разбита на отдельные модули, хранимые в различных файлах. Причем эти модули могут быть заранее откомпилированы и уже содержаться в, так называемых, объектных файлах (*.obj). В этом случае создается файл проекта программы (*.prj), в котором перечисляются главная программа, дополнительные модули и при необходимости библиотечные файлы (*.lib).

Пример:

В файле mytype.h содержатся описания типов и пр. Файл myfunc.cpp включает в себя некоторые функции, реализующие алгоритм решения задачи. Пусть этот файл предварительно откомпилировван и получен файл myfunc.obj. основная программа пусть содержится в файле myprog.cpp. Предположим, что используется некая созданная ранее нами библиотека mylib.lib. Тогда с помощью любого текстового редактора можно создать файл проекта myproject.prj вида

```
myprog.cpp
myfunc.obj
mylib.lib
```

Программное обеспечение имеет средства, позволяющие откомпилировать всю программу в целом по указанному проекту и создать выполняемый файл (*.exe). По умолчанию этот файл будет иметь имя проекта myproject.exe.

Литература

1. Болски М.И. Язык программирования Си.-М.: Радио и связь, 1988.-96 с.
2. Двоеглазов И.М. Язык программирования С++.-Киев: Евроиндекс, 1993.-128 с.
3. Страуструп Б. Развитие языка программирования С++.-Ногинск-5, Типография в-ч 20760, 1989.-286 с.
4. Лукас П. С++ под рукой.-Киев: НИПФ ДиаСофт, 1993.-176 с.
5. Дьюхарст С., Старк К. Программирование на С++.-Киев: НИПФ ДиаСофт, 1993.-272 с.
6. Вайнер Р., Пинсон Л. С++ изнутри.-Киев: ДиаСофт, 1993.-304 с.

Навчальне видання

КОРОТКИЙ ДОВІДНИК З МОВИ ПРОГРАМУВАННЯ С++

Методичні вказівки
для студентів I-IV курсів
спеціальності “Механіка”

Укладач ІЄВЛЕВ Іван Іванович

Відповідальний за випуск І. І. Ієвлев
Редактор І. Ю. Агаркова
Коректори: О. В. Гавриленко
О. В. Плахоніна

Редакційно-видавничу та додрукарську підготовку виконано організаційно-видавничим відділом НМЦ
61077, м. Харків, пл.Свободи, 4. Харківський національний університет ім. В. Н. Каразіна

Надруковано ПП Азамаєв В. Р.
61144, м. Харків, вул. Героїв Праці, 17

Підп.до друку 02.09.03. Формат 60x84 1/16.
Умовн.-друк.арк. 3,12. Облік.-вид.арк. 2,9.
Тираж 100 прим.