Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Ивановская государственная текстильная академия»
(ИГТА)

Кафедра прикладной математики и информационных технологий

Основы C++ - классы

УЧЕБНОЕ ПОСОБИЕ
по дисциплинам
«Технологии программирования»
«Прикладное программирование в информационных системах»,
«Программная реализация средств дизайна»
для студентов дневной формы обучения
специальности 740100
«Информационные технологии в дизайне»

Иваново 2013

Учебное пособие предназначено для студентов, изучающих основы программирования в C++ - ориентированной системе. Данное учебное пособие направлено так же на возможность приобретения некоторыми студентами базовых принципов решения задач по разработке программного обеспечения в C++ - ориентированной системе.

Учебное пособие является более углублённой методикой изучения основ C++ - использование классов по отношению к предшествующим учебным материалам, например, – Введение в C/C++.

Составитель:

Доцент кафедры ПМИТ, к.т.н., доцент Д. Д. Ветчинин

Научный редактор:

Заведующий кафедрой ПМИТ, д.т.н., профессор Н. А. Коробов

Введение

Так же, как и более ранние версии учебных материалов этого направления обучения, предложенные тем же автором /1, 2/, данный вариант учебного пособия направлен на изучение профессиональных подходов в технологиях программирования и программных реализациях программного обеспечения с использованием концепций и синтаксиса системы C++.

На этапе начала освоения материалов этого учебного пособия предполагается, что студент освоил синтаксис и основные программные конструкции (элементы) в C++ - системе /1/, в том числе, в их соотношении с другими системами программирования высокого уровня и, в соотношении с программированием на двух более низких уровнях.

Предполагается также, что обучающийся приобрёл начальные сведения по основной программной единице системы C++ - класс. Под начальными сведениями при этом следует понимать, прежде всего, основное «жизненное» назначение этой программной единицы – утилизация разработки прикладного программного обеспечения. К начальным же сведениям следует отнести базовые принципы построения классов и объектов классов: инкапсуляция, полиморфизм, наследование. Начальными же сведениями на этом уровне обучения являются: синтаксис C++ при работе с классами; дизайн исходного программного текста; основные реализационные принципы; программные единицы классов, варианты их реализации и манипуляции ими.

Отметим, что в сложившемся на настоящий момент формате учебного процесса, предлагаемая этим учебным материалом тематика, осваивается студентами параллельно с изучением основ работы по технологии MS VC++ /2/. Будем предполагать, что более углублённое представление о технологиях работы с классами «исходного» C++ поможет пониманию фрагментов программных текстов MS VC++ и частичному пониманию концепции в целом. Естественным должно представляться то, что предложенные в учебном пособии программные примеры реализованы в консольном варианте.

В заключении можно отметить – одна из базовых технологий C++ - наследование. Философски можно отметить - C++ как система также инкапсулировал эту технологию. Система C++ имеет множество производных систем (наследников): Java; C#; MFC; PHP; MCF; VC++ Builder; Objective C++; Macromedia Flash и др. Освоив исходные концепции и технологии, можно конкретизировать их и более актуальными на данный момент времени.

1. Перезагружаемые конструкторы

```
1   //Программа 1.1. Пример перегружаемого конструктора
2
3   #include <iostream>
4   #include <cstdlib>
5   //его поддключение и не нужно в консоли MS VS9! - почему?
6
7   #include <ctime>
8
9   using namespace std;
10
11  class timer{
12          int sec;
13
14  public:
15
16          timer(char *pt) {sec = atoi(pt);} // atoi() – классика C
17
18  //      Ctime(char *time) {seconds = _wtoi(time);} //MFC VS 2008
19  // в консоли - не проходит! - почему? – библиотека отключена!?
20
21          timer(int t) {sec = t;}
22
23          timer(int minute, int seconds) {sec = minute*60 + seconds;}
24
25          void run();
26  };
27
28  void timer::run()
29  {
30          clock_t t1;
31          t1 = clock();
32
33          while((clock()/CLOCKS_PER_SEC - t1/CLOCKS_PER_SEC) < sec);
34                  cout << sec << endl;
35                  cout << "\a";
36  }
37
38  void main()
39  {
40  //конструкторы - пример статической инициализации переменных
41          timer x(3), y("5"), z(0, 7);
42          x.run();
```

```
43          y.run();
44          z.run();
45
46  //конструкторы - пример динамической инициализации переменных
47          cout << "Enter seconds: " << endl;
48          char s[120];
49          cin >> s;
50          timer a(s);
51          a.run();
52
53          cout << "Enter minute's & seconds: ";
54          int min, sec;
55          cin >> min >> sec;
56          timer b(min, sec);
57          b.run();
58  }
```

Примечания: 1. <cstdlib> - заголовочный файл библиотеки, содержащей функции преобразования типов. 2. atoi() - функция преобразования данных типа char в тип int. 3. clock() – возвращает число тактов от системных часов от момента запуска программы. 4. clock_t – тип данных, разновидность длинного целого, данные этого типа возвращает функция clock().

2. Friend – функции класса

```
1   // Программа 2.1.  Пример использования Friend – функции класса
2   #include <iostream>
3   using namespace std;
4
5   const int ID = 0;
6   const int IN = 1;
7   class C2; //опережающее объявление класса C2
8
9   class C1 {
10          int mode; //ID – если сообщение не активно
11                    //IN – если сообщение выведено на экран
12  public:
13          void set(int);
14          friend int id(C1 a, C2 b);
15  };
16
17  class C2 {
18          int mode; //то же (ID и IN)
```

```
19  public:
20          void set(int);
21          friend int id(C1 a, C2 b);
22  };
23
24  void C1::set(int st) {
25          mode = st;
26  }
27
28  void C2::set(int st) {
29          mode = st;
30  }
31
32  //определение friend - функции классов C1, C2
33  int id(C1 a, C2 b)
34  {
35          if(a.mode || b.mode) return 0;
36          else return 1;
37  }
38
39  void main()
40  {
41          C1 x; C2 y;
42
43          x.set(ID);
44          y.set(ID);
45
46          if(id(x, y)) cout << "Screan fredom" << endl;
47          else cout << "Showing message" << endl;
48
49          x.set(IN);
50
51          if(id(x, y)) cout << "Screan fredom" << endl;
52          else cout << "Showing message" << endl;
53  }
```

Функция id() “дружественна” двум классам: C1 и C2 (имеет доступ к их закрытым членам). Результат - состояние объектов классов C1 и C2 можно проверить одним вызовом функции ID. В объявлении функции ID в классе C1 используется ссылка на класс C2 до его определения. «Дружественная» функция может быть определена и вне классов и в другом классе.

3. Динамическое выделение памяти в C++

```
1   // Программа 3.1. Динамическое выделение памяти для переменной
2   #include <iostream>
3
4   using namespace std;
5
6   int main()
7   {
8           int *p;
9           p = new int;
10          *p = 10;
11          int x = 10;
12
13          cout << "value: " << *p << endl;
14          cout << "address cell new int: " << p << endl;
15          cout << "address pointer p: " << &p << endl;
16          cout << "address variable x: " << &x << endl;
17
18          delete p;
19          return 0;
20  }
```

```
1   //Программа 3.2. Динамическое выделение памяти для одномерного
2   // массива
3
4   #include <iostream>
5   using namespace std;
6
7   int main()
8   {
9           double *p;
10          int i;
11          p = new double [10];
12
13          int x[10] = {0};
14
15          for(i = 0; i < 10; i++) p[i] = 3.54 + i;
16          cout << "values array of new douoble ";
17          for(i = 0; i < 10; i++) cout << p[i] << endl;
18
19          cout << "address pointer p: " << &p << endl;
20          cout << "address array x: " << x << endl;
```

```
21          cout << "address array to heap: " << p << endl;
22
23          delete [] p;
24
25          return 0;
26  }
```

```
1   /* Программа 3.3. Динамическое выделение памяти для двухмерного
2   массива. Используется двухуровневая косвенная адресация (указатель
3   на указатель). Организация двумерного динамического массива
4   производится в два этапа. Создаётся одномерный массив указателей,
5   затем каждому элементу этого массива присваивается адрес
6   одномерного массива. */
7
8   #include <iostream>
9   using namespace std;
10
11  void main() {
12          int i, j;
13          int dim1 = 5, dim2 = 2;
14
15          int **pAr2 = new int*[dim1];
16          for(i = 0; i < dim1; i++) pAr2[i] = new int[dim2];
17
18          for(j = 0; j < dim2; j++) {
19                  for(i = 0; i < dim1; i++) {
20                          if(j == 0)
21                                  pAr2[i][j] = i;
22                          else
23                                  pAr2[i][j] = i + 1;
24                  }
25          }
26
27            for(i = 0; i < dim1; i++) {
28                    for(j = 0; j < dim2; j++) {
29                            cout << pAr2[i][j] << " ";
30                    }
31                    cout << endl;
32            }
33
34          cout << "address variable i: " << &i << endl;
35          cout << "address array to heap: " << pAr2 << endl;
36
```

```
37          for (i = 0; i < dim1; i++) delete [] pAr2[i];
38          delete [] pAr2;
39  }
```

4. Указатель this

```
1   // Программа 4.1. Примеры манипуляций указателем this
2   // Только функции-члены класса имеют указатель this,
3   // на friend-функции он не распросраняется.
4
5   #include <iostream>
6   using namespace std;
7
8   class test_this{
9           int x;
10  public:
11          void set_x(int val) { this->x = val; }
12                  /* Реалистичный вариант инструкции присвоения
13                          значения переменной x. Сокращённая форма
14                          этой инструкции - x = val; (используется
15                          в большинстве случаев. */
16
17          void get_x();
18  };
19
20  void test_this::get_x()
21  {
22          cout << "       x = " << x << endl;
23          cout << "  this->x = " << this->x << endl;
24
25  // Операция "." и разыменованный this приведены  в учебных целях
26          cout << "(*this).x = " << ( *this ).x << endl;
27
28          cout << "Address pointer this class test_this - "
29                  << this << endl;
30  }
31
32  void main()
33  {
34          test_this ob1;
35
```

```
36          ob1.set_x(2017);
37          ob1.get_x();
38          cout << "Address object ob1 - " << & ob1 << endl;
39          cout << "Byte to object ob1 - " << sizeof ob1 << endl;
40
41          cout  << "Address main() - " << &main << endl;
42          cout  << "Address get_x() - " << &test_this::get_x << endl;
43          cout  << "Address set_x() - " << &test_this::set_x << endl;
44   }
```

5. Передача объектов функциям, возвращение объектов функциями

Объект класса можно передать функции аналогично объектам других типов. Передача осуществляется по значению – передаётся копия объекта.

```
1   // Программа 5.1. Пример передачи объекта функции в качестве
2   // аргумента (конструктор и деструктор копии).
3
4   #include <iostream>
5   using namespace std;
6
7   class obj7_2 {
8           int i;
9   public:
10          void in_i(int x) { i = x; }
11          void out_i() { cout << i << " " << endl; }
12  };
13
14  void fu(obj7_2 x)
15  {
16          x.out_i();   //
17          x.in_i(45); // устанавливает только локальную копию объекта
18          x.out_i();   //
19  }
20
21  int main()
22  {
23          obj7_2 o1;
24          o1.in_i(33);
25          fu(o1);
26          o1.out_i(); // изменений i не произошло
```

```
27
28  // Результат: 33 45 33
29
30          return 0;
31  }
```

```
1   /* Программа 5.2. Пример, демонстрирующий потенциальную
2     проблемму при передаче объекта функции (деструктор вызывается
3     два раза по одинаковому адресу). В примере приведён
4     и вариант решения подобной проблеммы. */
5
6   #include <iostream>
7   using namespace std;
8
9   class Cobj73 {
10          int *p;
11  public:
12          Cobj73(int x);
13          ~Cobj73();
14          int getVal() { return *p; }
15          void outVal() { cout << "x = " << *p << endl; }
16  };
17
18  Cobj73::Cobj73(int x)
19  {
20      // Выделение памяти, адресуемой указателем p
21          cout << "Put (make) the memory pointer p.\n";
22          p = new int;
23          *p = x;
24  }
25
26  Cobj73::~Cobj73() //(ОПАСНО!!!)
27  {
28      // Освобождение памяти, адресуемой указателем p
29          cout << "Free (clear) the memory pointer p.\n";
30          delete p;
31  }
32
33  // Выполнение этой функции – возникновение двух проблемм
34  void fu73(Cobj73 obC) // Runtime - two problems
35  {
36          cout << "x = " << obC.getVal() << endl;
```

```
37  }
38
39  // Этот вариант ф-ии не должен создавать проблемм. Объект
40  // передаётся по ссылке. Копия объекта не создаётся.
41  // Следовательно, объект не выходит из области видимости.
42
43  // void fu73(Cobj73 &obC) // No problems runtime
44  // {
45  //      cout << "x = " << obC.getVal() << endl;
46  // }
47  // Object to transmit by reference. Copy argumente – not
48  // found. Object(foundation) is region after utilization copy
49
50  void main()
51  {
52          Cobj73 obF(23);
53
54          fu73(obF); // Display x (obC ?)
55
56          obF.outVal(); // ??!!
57  }
```

```
1   // Программа 5.3. Пример, демонстрирующий возвращение объекта
2   // функцией
3
4   #include <iostream>
5   //#include <cstring>
6   using namespace std;
7
8   class Cobj74 {
9           char ls[33];
10  public:
11          void out_ls() { cout << ls << endl; }
12          void set_ls(char *p_ls) { strcpy(ls, p_ls); }
13  };
14
15  // Определение функции,возвращающей объект типа Cobj74
16  Cobj74 fu74()
17  {
18          char in_p_ls[33];
19          Cobj74 ob_ls;
20          cout << "Enter string: ";
21          cin >> in_p_ls;
```

```
22
23          ob_ls.set_ls(in_p_ls);
24
25          return ob_ls;
26  }
27
28  int main()
29  {
30          Cobj74 ob;
31
32          /* Присвоение данных объекта, возвращаемого ф-ей
33             fu74() объекту ob. */
34          ob = fu74();
35          ob.out_ls();
36
37          return 0;
38  }
```

```
1   // Программа 5.4. Демонстрация потенциальной проблемы при возврате
2   // функциями данных, предоставленных объектами (ОПАСНО!!!)
3
4   #include <iostream>
5   #include <cstring>
6   #include <cstdlib>
7   using namespace std;
8
9   class Cobj76 {
10          char *p_ls;
11  public:
12          Cobj76() { p_ls = 0; }
13          ~Cobj76() { if(p_ls) delete [] p_ls;
14                                  cout << "Free (clear) the memory pointer p_ls.\n"; }
15          void out_ls() { cout << p_ls << endl; }
16          void set_ls(char *);
17  };
18
19  // Определение ф-ии для выделения памяти, адресуемой указателем p_ls
20  // и заполнения ячеек heap значениями строкового литерала
21  //
22  void Cobj76::set_ls(char *p_str)
23  {
24          p_ls = new char[strlen(p_str) + 1];
25          strcpy(p_ls, p_str);
```

```
26  }
27
28  // Определение ф-ии
29  //
30  Cobj76 fu76()
31  {
32          char in_str[33];
33          Cobj76 p_str;
34
35          cout << "Enter string: ";
36          cin >> in_str;
37
38          p_str.set_ls(in_str);
39          return p_str;
40  }
41
42
43  void main()
44  {
45          Cobj76 obF;
46          /* Присвоение объекта, возвращаемого ф-ей fu76()
47             объекту obF */
48          obF = fu76(); // Генерация ошибки д.б.
49
50          obF.out_ls(); // д.б. отображение "мусора".
51  }
```

```
1   // Программа 5.5. Пример использования конструктора копии при
2   // передаче функции объекта в качестве параметра, конструктор
3   // копии – корректное побитное копирование блока памяти для объекта
4   // в блок памяти с индивидуальным адресом.
5
6
7   #include <iostream>
8   #include <cstdlib>
9   using namespace std;
10
11  class Cobj77 {
12          int *p;
13  public:
14          Cobj77(int i);                  // "обычный" конструктор
15          Cobj77(const Cobj77 &ob);       // конструктор копии
16          ~Cobj77();
```

```
17          int getVal() { return *p; }
18  };
19
20  // Определение "обычного" конструктора
21  Cobj77::Cobj77(int i)
22  {
23          cout << "Put (make) the memory pointer p.\n";
24          p = new int;
25          *p = i;
26  }
27
28  // Определение конструктора копии
29  Cobj77::Cobj77(const Cobj77 &obj)
30  {
31          p = new int;
32          *p = *obj.p; //устанавливается значение копии объекта
33          cout << "Call constructor of copy.\n";
34  }
35
36  Cobj77::~Cobj77()
37  {
38          cout << "Free (clear) the memory pointer p.\n";
39          delete p;
40  }
41
42  // Определение ф-ии, принимающей объект-параметр
43  void out(Cobj77 ob)
44  {
45          cout << ob.getVal() << '\n';
46  }
47
48  void main()
49  {
50          Cobj77 obF(77);
51
52          out(obF);
53  }
```

```
1   // Программа 5.6. Пример использования конструктора копии при
2   // инициализации одного объекта другим объектом
3
4   #include <iostream>
5   #include <cstdlib>
```

```
6   using namespace std;
7
8   class Cobj78 {
9           int *p;
10  public:
11          Cobj78(int i);                  // "обычный" конструктор
12          Cobj78(const Cobj78 &ob);       // конструктор копии
13          ~Cobj78();
14          int getVal() { return *p; }
15  };
16
17  // Определение "обычного" конструктора
18  Cobj78::Cobj78(int i)
19  {
20          cout << "Put the memory pointer p by ordinary constructor.\n";
21          p = new int;
22          *p = i;
23  }
24
25  // Определение конструктора копии
26  Cobj78::Cobj78(const Cobj78 &ob)
27  {
28          p = new int;
29          *p = *ob.p;  //устанавливается значение копии объекта
30          cout << "Put the memory pointer p by constructor of copy.\n";
31  }
32
33  Cobj78::~Cobj78()
34  {
35          cout << "p = " << p << " - Free (clear) the memory pointer p.\n";
36          delete p;
37  }
38
39  void main()
40  {
41          Cobj78 ob1(81);      // Вызов "обычного" конструктора
42
43          Cobj78 ob2 = ob1;  // Вызов конструктора копии
44  }
```

```
1   // Программа 5.7. Пример - Вызов конструктора копии при создании
2   // временного бъекта в качестве значений, возвращаемого функцией
3   // объекта
4
5   #include <iostream>
6   using namespace std;
7
8   class Cobj79 {
9   public:
10          Cobj79() { cout << "Ordinary constructor.\n"; }
11          Cobj79(const Cobj79 &object) {
12                  cout << "Constructor of copy.\n"; }
13  };
14
15  // Определение ф-ии, возвращающую объект
16  Cobj79 fu79()
17  {
18          Cobj79 ob; // Вызов "обычного" конструктора
19          return ob; // Неявный вызов конструктора копии
20  }
21
22  void main()
23  {
24          Cobj79 ob1; // Вызов "обычного" конструктора
25
26          ob1 = fu79(); // Вызов конструктора копии
27  }
```

6. Наследование, базовые понятия

```
1   // Программа 6.1. Демонстрационный пример использования
2   // технологии наследования для программной реализации фрагмента
3   // предметной области – член сообщества ВУЗа.
4
5   #include <iostream>
6   using namespace std;
7
8   class Member{
9   protected:
10          char name[33];
11          int length;
```

```
12  public:
13          void set_Name() {
14                  cout << "Enter name" << endl;
15                  cin >> name;
16                  cout << '\n';
17          }
18
19          void set_length(int num) {length = num;}
20          int get_length() {return length;}
21          void show();
22  };
23
24          void Member::show()  {
25                  cout << name << "\n - length of work - " << length
26                          << endl << endl;
27          }
28
29
30  class Student : public Member {
31  public:
32          void show() {
33                  cout << "Student - " << name << "\n - cours - "
34                          << length << endl << endl;
35          }
36  };
37
38
39  class Teacher : public Member {
40          int teacher;
41  public:
42          void set_teacher (int num_t) {teacher = num_t;}
43          int get_teacher () {return teacher;}
44
45          void show_T() {
46                  cout << " - length of teacher - " << teacher << endl
47                          << endl;
48          }
49  };
50
51  class Magister : public Student {
52  public:
53          void show() {
54                  cout << "Magister - " << name << "\n - cours - "
55                          << length << endl << endl;
56          }
```

```
};

void main(){
        Teacher t1;
        Student s1;
        Magister m1;

        t1.set_Name();
        s1.set_Name();
        m1.set_Name();

        t1.set_length(30);
        t1.set_teacher(22);

        s1.set_length(4);
        m1.set_length(6);

        t1.show();
        t1.show_T();

        s1.show();
        m1.show();
}
```

```
// Программа 6.2. Учебный пример наследования двух
// базовых классов

#include <iostream>
using std::cout;

class foundX {
protected:
    int x;
public:
   void outX(){ cout << x << "\n"; }
};

class  foundY {
protected:
   int y;
public:
        void outY() { cout << y << "\n"; }
};
```

```
20
21  class heir: public foundX, public foundY {
22  public:
23          void setXY(int k, int l) { x = k; y = l;}
24  };
25
26  int main()
27  {
28          heir ob;
29          ob.setXY(11, 23);
30          ob.outX();
31          ob.outY();
32
33      return 0;
34  }
```

```
1   // Программа 6.3. Демонстрационный пример передачи параметров
2   // конструктором производного класса конструктору(ам) базового(ых)
3   // класса(ов), порядок выполнения конструкторов/деструкторов
4   // в иерархии наследования.
5
6   #include <iostream>
7   using std::cout;
8
9   class found1 {
10  protected:
11      int x1;
12  public:
13          // Creation - создание; specimen - образец;
14          found1(int x) { x1 = x; cout << "Creation found1 - object.\n"; }
15          ~found1() { cout << "Destruction found1 - objectn.\n"; }
16  };
17
18  class found2 {
19  protected:
20      int x2;
21  public:
22          found2(int x) { x2 = x; cout << "Creation found2 - object.\n"; }
23          ~found2() {cout << "Destruction found2 - object.\n"; }
24  };
25
26  // класс heir1 "открыто" наследует два базовых класса
27  class heir1: public found1, public found2 {
```

```
28          int x3;
29  public:
30  // параметризированный конструктор класса heir1 при вызове
31  // передаёт параметры (аргументы) конструкторам базовых классов
32     heir1(int x, int y, int z): found1(y), found2(z)
33                     { x3 = x; cout << "Creation heir1 - object.\n";}
34     ~heir1() { cout << "Destruction heir1 - object.\n";}
35          void out() {cout << x1 << " " << x2 << " " << x3 << "\n";}
36  };
37
38  int main ()
39  {
40    heir1 sp (22, 33, 48);
41  //   heir2();
42     sp.out() ;
43
44    return 0 ;
45  }
```

```
1     // Программа 6.4. Учебный пример реализации фрагмента предметной
2     // области – «ВУЗ – его люди» с использованием технологий C/C++
3     // классы, наследование в иерархии. В примере используется
4     // «профессиональный стиль» - организация классов в формате
5     // заголовочных файлов и файлов реализации.
6
7     //Заголовочный файл класса Member - member.h
8
9     #ifndef MEMBER_H
10    #define MEMBER_H
11
12    class Member{
13    protected:
14            char *name;
15                    int length;
16    public:
17            Member(const char*);
18            ~Member();
19            void set_length(int num) {length = num;}
20            int get_length(){return length;}
21            void show() const;
22    };
23    #endif
24
```

```
25
26  // Заголовочный файл класса Student - student.h
27
28  #include "member.h"
29  #include <iostream>
30  using namespace std;
31
32  class Student : public Member {
33  public:
34          Student(const char *fcn) : Member(fcn){}
35
36          void Student::show()
37          {cout << name << " - course - " << length << endl << endl;}
38  };
39
40
41  // Заголовочный файл класса Teacher - teacher.h
42
43  #include "member.h"
44
45  class Teacher: public Member {
46          int teacher;
47  public:
48          Teacher(const char *fcn) : Member(fcn){}
49          void set_teacher (int num_t) {teacher = num_t;}
50          int get_teacher () {return teacher;}
51          void show_t () const;
52  };
53
54
55  // Заголовочный файл класса Worker - worker.h
56
57  #include "member.h"
58  #include <iostream>
59  using namespace std;
60
61  class Worker : public Member {
62  public:
63          Worker(const char *fcn) : Member(fcn){}
64  };
65
66
67  // Файл реализации класса Member - member.cpp
68
69  #include <iostream>
```

```
using namespace std;
#include <cstring>
#include <cassert>
#include "member.h"

Member::Member(const char *fcn)
{
        name = new char[strlen(fcn) + 1];
        assert(name != 0);
        strcpy(name, fcn);
}
Member::~Member()
{
        delete [] name;
}
void Member::show() const {
        cout << name << " - length of work - " << length << endl
                << endl;}


// Файл реализации класса Teacher - teacher.cpp

#include "teacher.h"
#include <iostream>
using namespace std;

void Teacher::show_t()const //????
{cout << name << " - length of teacher - " << teacher << endl
        << endl;}


// Реализация клиента заданных классов

#include "teacher.h"
#include "student.h"
#include "worker.h"

void main()
{
        Teacher t1 ("Muhamed Hadgar");
        Student s1("Petr Ivanov");
        Worker w1("Zina Ulich");

        t1.set_length(15);
        t1.set_teacher(9);
```

```
115         t1.show();
116         t1.show_t();
117
118         s1.set_length(4);
119         s1.show();
120
121         w1.set_length(11);
122         w1.show();
123  }
```

```
1   // Программа 6.5. Аналог программs 6.4. Учебный вариант – отсутствует
    // разделение на «заголовки» и «реализацию».
2
3   #include <iostream>
4   using namespace std;
5   #include <cstring>
6   #include <cassert>
7
8   class Member{
9   protected:
10          char *name;
11                  int length;
12  public:
13          Member(const char*); //
14          ~Member();
15          void set_length(int num) {length = num;} //
16          int get_length(){return length;} //
17          void show() const; //
18  };
19
20  Member::Member(const char *fcn) //
21  {
22          name = new char[strlen(fcn) + 1];
23          assert(name != 0);
24          strcpy(name, fcn);
25  }
26  Member::~Member()
27  {
28          delete [] name;
29  }
30  void Member::show() const {
31          cout << name << " - length of work - " << length << endl
32                  << endl;}
```

```
33
34
35  class Teacher: public Member {
36          int teacher;
37  public:
38          Teacher(const char *fcn) : Member(fcn){}
39          void set_teacher (int num_t) {teacher = num_t;}
40          int get_teacher () {return teacher;}
41          void show_t () const;
42  };
43
44  void Teacher::show_t()const //
45  {cout << name << " - length of teacher - " << teacher << endl
46          << endl;}
47
48
49  class Student : public Member {
50  public:
51          Student(const char *fcn) : Member(fcn){}
52
53          void Student::show() //
54          {cout << name << " - course - " << length << endl << endl;}
55  };
56
57
58  class Worker : public Member {
59  public:
60          Worker(const char *fcn) : Member(fcn){}
61  };
62
63  void main()
64  {
65          Teacher t1 ("Muhamed Hadgar");
66          Student s1("Petr Ivanov");
67          Worker w1("Zina Ulich");
68
69          t1.set_length(15);
70          t1.set_teacher(9);
71          t1.show();
72          t1.show_t();
73
74          s1.set_length(4);
75          s1.show();
76
77          w1.set_length(11);
```

```
78          w1.show();
79  }


1   /* Программа 6.6. Демонстрация предоставления доступа к членам
2      класса при наследовании - стандарт ANSI/ISO – тип наследования.
3     некоторые разработчики считают более корректным стилем
4     предоставления такого доступа - использование пространств имён.
5   */
6   #include <iostream>
7   using std::cout;
8
9   class found {
10     int x;
11  public:
12          int y, z;
13     void setX(int x1) { x = x1; }
14          int getX() { return x; }
15  };
16
17  class heir: private found {
18  public:
19          /* Команды переопределения private-наследования
20             класса found - восстановление public доступа
21             для трёх его членов. НЕВОЗМОЖНО! - для private
22             членов базового класса.
23          */
24
25          found::y;
26          found::getX;
27          found::setX;
28
29          int g;
30  };
31
32  int main()
33  {
34          heir ob;
35          //error C2248: 'x' : cannot access private member
36          //declared in class 'found'
37
38          //ob.x = 33;
39
40          ob.y = 33;
41
```

```
42          //error C2248: 'z' : cannot access public member
43          //declared in class 'found'
44
45          //ob.z = 33;
46
47          ob.g = 41;
48          ob.setX(45);
49
50          cout << ob.getX() << " " << ob.y << " " << ob.g << '\n';
51
52      return 0;
53  }
```

```
1   /* Программа 6.7. Учебный пример (т.е., "тупой") использования
2      собственного пространства имён по примеру
3      востановления доступа к членам базового класса.
4   */
5   #include <iostream>
6   using namespace std;
7
8   namespace mnd {
9
10  class found {
11      int x;
12  public:
13          int y, z;
14      void setX(int x1) { x = x1; }
15          int getX() { return x; }
16  };
17                          }
18
19  //using namespace mnd; // можно было бы и так - работает
20
21  class heir: private mnd::found {
22  public:
23          /* Команды переопределения private-наследования
24             класса found - восстановление public доступа
25             для трёх его членов. НЕВОЗМОЖНО! - для private
26             членов базового класса. Использование VC6
27             особых отличий "нового" от "старого" - не показало.
28          */
29
30          //error C2602: 'mnd::found::y' is not a member
```

```
31          //of a base class of 'heir'
32
33          ///using mnd::found::y;
34
35          found::getX;
36          found::setX;
37          found::y;
38
39          int g;
40  };
41
42  int main()
43  {
44          heir ob;
45
46          //error C2248: 'x' : cannot access private member
47          //declared in class 'found'
48
49          //ob.x = 33;
50
51          ob.y = 33;
52
53          //error C2248: 'z' : cannot access public member
54          //declared in class 'found'
55
56          //ob.z = 33;
57
58          ob.g = 41;
59          ob.setX(45);
60
61          cout << ob.getX() << " " << ob.y << " " << ob.g << '\n';
62
63      return 0;
64  }
```

7. Виртуальные классы

```
1   // Программа 7.1. Программа не должна откомпилироваться из-за
2   // двусмысленности при обращении к объекту
3   // (ambiguous – двусмысленный).
4
5   #include <iostream>
6   using std::cout;
7
8   class found {
9   public :
10          int x;
11  };
12
13  class hear1: public found {
14  public:
15          int y;
16  };
17
18  class hear2: public found {
19  public:
20          int z;
21  };
22
23  class hearHear: public hear1, public hear2 {
24          public:
25                  int mult;
26  };
27
28  int main()
29  {
30          hearHear ob;
31          ob.x = 47; //error C2385: 'hearHear::x' is ambiguous
32                                  //warning C4385: could be the 'x' in base
33                                  //'found' of base 'hear1' of class 'hearHear'
34                                  //or the 'x' in base 'found' of base 'hear2'
35                                  //of class 'hearHear'
36
37          ob.y = 32;
38          ob.z = 23;
39
40          ob.mult = ob.x * ob.y * ob.z; //error C2385, warning C4385
41
```

```
42          cout << ob.x <<  //error C2385, warning C4385
43                  " " << ob.y << " " << ob.z << " ";
44
45          cout << sp.mult << "/n";
46  }
```

```
1   // Программа 7.2. Пример использования оператора выбора контекста
2   // (разрешения области видимости) для устранения
3   // неопределённости выбора члена класса.
4
5   #include <iostream>
6   using namespace std;
7
8   class found {
9   public :
10          int x;
11  };
12
13  class hear1: public found {
14  public:
15          int y;
16  };
17
18  class hear2: public found {
19  public:
20          int z;
21  };
22
23  class hearHear: public hear1, public hear2 {
24          public:
25                  int mult;
26  };
27
28  int main()
29  {
30          hearHear ob;
31
32          ob.hear1::x = 9; // Неопределённости нет
33
34          ob.hear2::x = 3; // Неопределённости нет
35
36          ob.y = 2;
37          ob.z = 7;
```

```
38
39          ob.mult = ob.hear1::x * ob.hear2::x * ob.y * ob.z;
40
41          cout << ob.hear1::x << " " << ob.hear2::x << " "
42
43                  << ob.y << " " <<ob.z << " "<< ob.mult << endl;
44
45          return 0;
46  }
```

```
1   // Программа 7.3. Пример использования виртуального базового класса
2   // для устранения неопределённости выбора члена
3   // класса при множественном наследовании.
4   // Копия виртуального класса - только одна.
5   // Производные классы эту копию "не размножают".
6
7   #include <iostream>
8   using namespace std;
9
10  class found {
11  public:
12          int x;
13  };
14
15  // Наследование found как виртуального класса
16
17  class hear1 : virtual public found {
18  public:
19          int y;
20  };
21
22  class hear2 : virtual public found {
23  public:
24          int z;
25  };
26
27  // Результат - класс hearHear содержит одну копию found
28
29  class hearHear: public hear1, public hear2 {
30  public:
31          int mult;
32  };
33
```

```
34   int main()
35   {
36           hearHear ob;
37
38           ob.x = 11; // Неоднозначности нет
39           ob.y = 2;
40           ob.z = 10;
41
42           ob.mult = ob.x * ob.y * ob.z;
43
44           cout << ob.x << " " << ob.y << " " << ob.z
45                   << " "<< ob.mult << endl;
46
47           return 0;
48   }
```

```
1    // Программа 7.4. Пример использования указателя на базовый тип
2    // (базовый класс - БК). Тема - виртуальные функции, динамический
3    // полиморфизм.
4
5    #include <iostream>
6    #include <cstring>
7    using namespace std;
8
9    class found {
10           char teacher[33];
11   public:
12           in_name_T(char *t) { strcpy(teacher, t); }
13           out_name_T() { cout << teacher << endl; }
14   };
15
16   class hear : public found {
17           char diplomant[33];
18   public:
19           in_name_D(char *d) { strcpy(diplomant, d); }
20           out_name_D() { cout << "Diplomant: "
21                   << diplomant << endl; }
22   };
23
24   void main()
25   {
26           found *p_fnd;
27           found ob_fnd;
```

```
28
29          hear *p_her;
30          hear ob_her;
31
32          p_fnd = &ob_fnd; // Указателю БК присваивается адрес
33                                          // объекта базового класса
34          p_fnd->in_name_T("Corobov N. A."); // Доступ через указатель БК
35                                                          // к объекту БК
36          p_fnd = &ob_her; // Указателю БК присваивается адрес
37                                          // объекта производного класса
38          p_fnd->in_name_T("Egorova N. A."); // Доступ через указатель БК
39                                                          //  к объекту ПК
40
41          ob_fnd.out_name_T(); // объект базового класса
42          ob_her.out_name_T(); // объект ПК
43          cout << endl;
44
45          /* Доопределённые в производном классе ф-ии недоступны
46            через указатель на БК. Их вызов можно задать либо
47            непосредственно ( ob_her.out_name_D() ), либо через
48            указатель на ПК.
49          */
50
51          p_her = &ob_her;
52          p_her->in_name_D("Ivanov X. P.");
53
54          p_fnd->out_name_T(); // м.б.б. и через p_her
55          p_her->out_name_D();
56          cout << endl;
57
58          p_fnd->out_name_T(); // м.б.б. и через p_her
59          ((hear *)p_fnd)->out_name_D(); // можно и так?
60
61  }
```

```
1   // Программа 7.5. Пример виртуальной функции - пример 2,
2   // манипуляция виртуальной функцией с переопределением.
3
4   #include <iostream>
5   using namespace std;
6
7   class found {
8   public:
```

```
9           virtual void whatIs() { // объявление виртуальной функции
10                  cout << "Foundation class.\n\n";
11          }
12  };
13
14  class heir_first : public found {
15  public:
16          void whatIs() { // Переопределение функции whatIs для
17                                      // класса heir_first
18                  cout << "First derived class.\n\n";
19          }
20  };
21
22  class heir_second : public found {
23  public:
24          void whatIs() { // Переопределение функции whatIs() для
25                                      // класса heir_second.
26                  cout << "Second derived class.\n\n";
27          }
28  };
29
30  void main ()
31  {
32          found ob_found;
33          found *p_found;
34
35          heir_first ob_heir1;
36          heir_second ob_heir2;
37
38          p_found = &ob_found;
39          p_found->whatIs(); // вызов функции whatIs() класса found
40
41          p_found = &ob_heir1;
42          p_found->whatIs(); // вызов функции whatIs() класса ob_heir1
43
44          p_found = &ob_heir2;
45          p_found->whatIs(); // вызов функции whatIs() класса ob_heir2
46
47  }
```

```
1   // Программа 7.6. Виртуальные функции - пример 3, использование
2   // виртуальной функцией без переопределения.
3
4   #include <iostream>
5   using namespace std;
6
7   class found {
8   public:
9           virtual void whatIs() {
10                  cout << "Foundation class.\n\n";
11          }
12  };
13
14  class heir_first : public found {
15  public:
16          void whatIs() {
17                  cout << "First derived class.\n\n";
18          }
19  };
20
21  class heir_second : public found {
22          // Функция whatIs() в классе не переопределена.
23  };
24
25  int main()
26  {
27          found ob_found;
28          found *p_found;
29
30          heir_first ob_heir1;
31          heir_second ob_heir2;
32
33          p_found = &ob_found;
34          p_found->whatIs(); // доступ к функции whatIs() класса found
35
36          p_found = &ob_heir1;
37          p_found->whatIs(); // доступ к функции whatIs() класса heir_first
38
39          p_found = &ob_heir2;
40          p_found->whatIs(); /* Здесь выполняется обращение к функции
    whatIs()
41                                  класса found, поскольку в классе second_d
42                                  она не переопределена. */
43
```

```
44          return 0;
45  }
```

```
1   // Программа 7.7. Виртуальные функции - пример 4, виртуальная
2   // функция в иерархии наследования.
3
4   #include <iostream>
5   using namespace std;
6
7   class found  {
8   public:
9           virtual void whatIs() {
10                  cout << "Foundation class.\n\n";
11          }
12  };
13
14  class   heir_first : public found  {
15  public:
16          void  whatIs() {
17                  cout << "First derived class.\n\n";
18          }
19  };
20
21  // Класс heir_second выведен из класса heir_first,
22  // а не из класса found
23  class heir_second : public heir_first {
24          // Функция whatIs() не переопределена
25  };
26
27  int main ()
28  {
29          found ob_found;
30          found *p_found;
31
32          heir_first ob_heir1;
33          heir_second ob_heir2;
34
35          p_found = &ob_found;
36          p_found->whatIs(); // Вызов функции whatIs() класса found
37
38          p_found = &ob_heir1;
39          p_found->whatIs(); // Вызов функции whatIs() класса heir_first
40
```

```
41          p_found = &ob_heir2;
42          p_found->whatIs(); /* Выполняется обращение к функции
43                                    whatIs()    класса heir_first, поскольку в
44                                    классе heir_second она не переопределена,
45                                  и класс heir_first ближе по иерархии
    наследования*/
46  }
```

```
1   // Программа 7.8. Виртуальные функции – учебный пример
2   // программы, использующей виртуальную функцию -
3   // вычисление площадей 2d фигур.
4
5   #include   <iostream>
6   using  namespace   std;
7
8   class shape {
9   protected:
10          float a, b;
11  public:
12          void set_In(double x, double y = 0) {
13                  a = x;
14                  b = y;
15          }
16          virtual void out_area() {
17                  cout << "For this class expression to culculate";
18                  cout << "at area not definition.\n";
19          }
20  } ;
21
22  class triangle : public shape {
23  public:
24          void out_area() {
25                  cout << "Triangle thih a = ";
26                  cout << a << " and foundation b = " << b;
27                  cout << " is area ";
28                  cout << a * b * 0.5  << ".\n\n";
29          }
30  };
31
32  class rectangle : public shape {
33  public:
34          void out_area() {
35                  cout << "Rectangle thih sise ";
```

```
36                  cout << " a = " << a;
37                  cout << ", b = " << b;
38                  cout << " is area ";
39                  cout << a * b << ".\n\n";
40          }
41  };
42
43  class circle : public shape {
44  public:
45          void out_area() {
46                  cout << "Girсle thih radius ";
47                  cout << " r = " << a;
48                  cout << " is area ";
49                  cout << 3.14 * a * a << ".\n\n";
50          }
51  } ;
52
53  int main()
54  {
55          shape *p_shape; // создание указателя на базовый тип
56
57          triangle t;     // создание объектов
58          rectangle r;     // производных типов
59          circle c;
60
61          p_shape = &t;
62          p_shape->set_In(9.99, 1.790);
63          p_shape->out_area();
64
65          p_shape = &r;
66          p_shape->set_In(7.09,1.9);
67          p_shape->out_area();
68
69          p_shape = &c;
70          p_shape->set_In(9.1);
71          p_shape->out_area();
72          return 0;
73  }
```

```
1   // Программа 7.9. Учебный программный эскиз по теме:
2   // «чисто виртуальные функции» - абстрактный класс.
3
4   /*
5   Эта программа не должна скомпилироваться, поскольку в классе
6   circle нет переопределения функции out_area().
7   */
8
9   #include   <iostream>
10  using  namespace   std;
11
12  class shape {
13  protected:
14          float a, b;
15  public:
16          void set_In(double x, double y = 0) {
17                  a = x;
18                  b = y;
19          }
20          virtual void out_area() = 0; // чисто виртуальная функция
21  } ;
22
23  class triangle : public shape {
24  public:
25          void out_area() {
26                  cout << "Triangle thih a = ";
27                  cout << a << " and foundation b = " << b;
28                  cout << " is area ";
29                  cout << a * b * 0.5  << ".\n\n";
30          }
31  };
32
33  class rectangle : public shape {
34  public:
35          void out_area() {
36                  cout << "Rectangle thih sise ";
37                  cout << " a = " << a;
38                  cout << ", b = " << b;
39                  cout << " is area ";
40                  cout << a * b << ".\n\n";
41          }
42  };
43
44
```

```
45  // отсутствие определения ф-ии out-area - error
46  class circle : public shape {
47  public:
48          void out_area() {
49                  cout << "Gircle thih radius ";
50                  cout << " r = " << a;
51                  cout << " is area ";
52                  cout << 3.14 * a * a << ".\n\n";
53
54  /* C:\~…\cl_336\virtF5.cpp(62) : error C2259: 'circle' : cannot instantiate
55     abstract class due to following members:
56     C:\~…\cl_336\virtF5.cpp(45) : see declaration of 'circle'
57     C:\~…\cl_336\virtF5.cpp(62) : warning C4259: 'void __thiscall
58     shape::out_area(void)' : pure virtual function was not defined
59     C:\~…\cl_336\virtF5.cpp(21) : see declaration of 'out_area'
60     C:\~…\cl_336\virtF5.cpp(62) : error C2259: 'circle' : cannot instantiate
61     abstract class due to following members:
62     C:\~…\cl_336\virtF5.cpp(45) : see declaration of 'circle'
63     C:\~…\cl_336\virtF5.cpp(62) : warning C4259: 'void __thiscall
64     shape::out_area(void)' : pure virtual function was not defined
65     C:\~…\cl_336\virtF5.cpp(21) : see declaration of 'out_area'
66     Error executing cl.exe.
67  }
68  */
69  } ;
70
71  int main()
72  {
73          shape *p_shape; // создание указателя на базовый тип
74
75          triangle t;          // создание объектов
76          rectangle r;         // производных классов
77
78          circle c;            // error - создание этого объекта невозможно
79
80          p_shape = &t;
81          p_shape->set_In(9.99, 1.790);
82          p_shape->out_area();
83
84          p_shape = &r;
85          p_shape->set_In(7.09,1.9);
86          p_shape->out_area();
87
88          p_shape = &c;
89          p_shape->set_In(9.1);
```

```
90          p_shape->out_area();
91          return 0;
92  }
```

```
1   // Программа 7.10. - Пример реализации задачи программы 7.9. с
2   // использованием «чисто виртуальных функций» - абстрактного класса.
3   // Идиомы: «истинный полиморфизм» - «пуризм».
4
5   #include <iostream>
6   using namespace std;
7
8   class shape {
9   protected:
10          float a, b;
11  public:
12          void set_In(double x, double y = 0) {
13                  a = x;
14                  b = y;
15          }
16          virtual void out_area() = 0; // чисто виртуальная функция
17  };
18
19  class triangle : public shape {
20  public:
21          void out_area() {
22                  cout << "Triangle thih a = ";
23                  cout << a << " and foundation b = " << b;
24                  cout << " is area ";
25                  cout << a * b * 0.5  << ".\n\n";
26          }
27  };
28
29  class rectangle : public shape {
30  public:
31          void out_area() {
32                  cout << "Rectangle thih sise ";
33                  cout << " a = " << a;
34                  cout << ", b = " << b;
35                  cout << " is area ";
36                  cout << a * b << ".\n\n";
37          }
38  };
39
```

```
40  class circle : public shape {
41  public:
42          void out_area() {
43                  cout << "Girсle thih radius ";
44                  cout << " r = " << a;
45                  cout << " is area ";
46                  cout << 3.14 * a * a << ".\n\n";
47          }
48  };
49
50  int main()
51  {
52          shape *p_shape;     // создание указателя на базовый тип
53
54          triangle t;             // создание объектов
55          rectangle r;            // производных классов
56          circle c;
57
58          p_shape = &t;
59          p_shape->set_In(9.99, 1.790);
60          p_shape->out_area();
61
62          p_shape = &r;
63          p_shape->set_In(7.09,1.9);
64          p_shape->out_area();
65
66          p_shape = &c;
67          p_shape->set_In(9.1);
68          p_shape->out_area();
69          return 0;
70  }
```

8. Обработка исключений

```
1   /* Программа 8.1. /H. Schildt/ - "Простой пример обработки
2      исключений". Пример показывает, как организовать исключение -
3      точнее выброс из нормального хода выполнения программы с
4      последующим перехватом и обработкой этого выброса. К обработке
5      ошибки пример никакого отношения не имеет (ошибка - выброс
6      генерируются искусственно). В реальном программировании
```

```
7       использовать можно, но не надо бы.
8   */
9   #include <iostream>
10  using namespace std;
11
12  void main()
13  {
14          cout << "START\n";
15
16          try
17          {
18                  cout << "In try - bloc\n";
19                  throw 123; // искуственное генерирование ошибки - выброс
20                  cout << "This bloce code is not run.\n";
21                  cout << "In this point - control is not return";
22          }
23          catch (int i) // Перехват исключения
24          {
25                  cout << "To entercept at exception. This value is: ";
26                  cout << i << '\n';
27          }
28
29          cout << "END\n";
30  }
```

```
1   /* Программа 8.2. /H. Schildt/. По сути, пример является
2      аналогом предыдущего примера (8.1.), причем с аналогичными
3      замечаниями. Программа демонстрирует "не перехват" исключения.
4   */
5   #include <iostream>
6   using namespace std;
7
8   void main()
9   {
10          cout << "START\n";
11
12          try
13          {
14                  cout << "In try - bloc\n";
15                  throw 123; // генерирование выброса
16                  cout << "This bloce code is not run.\n";
17                  cout << "In this point - control is not return";
18          }
```

```
19          catch (float i) // Перехват исключения типа float
20          {
21                  cout << "To entercept at exception. This value is: ";
22                  cout << i << '\n';
23          }
24
25          cout << "END\n";
26  }
```

Примечание: Программа 8.2. должна сгенерировать Debug Error – abnormal program termination.

```
1   /* Программа 8.3. /H. Schildt/. Замечания по поводу "не перехвата"
2           ошибки - те же. Демонстрирует перехват выброса,
3           сгенерированного функцией, вызванной из тестового
4           блока этим же try - блоком.
5   */
6
7   #include <iostream>
8   using namespace std;
9
10  void funcTest(int X)
11  {
12          cout << "In function funcTest() - value X is: "
13                  << X << "\n";
14          if(X) throw X;
15  }
16
17  void main()
18  {
19          cout << "START\n";
20
21          try     {
22                  cout << "In try - bloc\n";
23                  funcTest(0);
24                  funcTest(1);
25                  funcTest(2);
26          }
27          catch (int i) // Перехват исключения типа int
28          {
29                  cout << "To entercept at exception. This value is: ";
30                  cout << i << '\n';
```

```
31          }
32
33          cout << "END\n";
34  }
```

```
1   /* Программа 8.4. /H. Schildt/. Замечания по поводу "не перехвата"
2           ошибки - те же. Демонстрирует локализацию try - блока
3           в функции. При каждом вызове такой функции запускается
4           обработка исключений в этой функции.
5           В цикле было бы то же самое (To entercept – перехватить).
6   */
7
8   #include <iostream>
9   using namespace std;
10
11  void funcTest(int X)
12  {
13          try     {
14                  if(X) throw X;
15          }
16          catch (int i) {
17                  cout << "Entercept! Exception's number : "
18                          << i << "\n";
19          }
20  }
21
22  void main()
23  {
24          cout << "START\n";
25
26          funcTest(1);
27          funcTest(2);
28          funcTest(0);
29          funcTest(3);
30
31          cout << "END\n";
32  }
```

```
1   /* Программа 8.5. Пример применения исключения.
2           Devide By Null Exception.
3      (Occurred - случилось, поимело место;
4       attempt - попытка; message –сообщение).
5           Конструктор - после ":" - инициализатор элемента
6           создаваемого объекта. Передаётся в конструкторы
7           элементов-объектов.
8   */
9
10  #include <iostream>
11  using namespace std;
12
13  class DBNE {
14  public:
15          DBNE() : msg("Attempted to devide by null") {}
16
17          char* whatIs() {return msg;}
18
19  private:
20          char *msg;
21  };
22
23  float funcDE( int a, int b)
24  {
25          if ( b == 0 ) throw DBNE();
26
27          return static_cast< float > (a ) / b;
28  }
29
30  void main()
31  {
32          int a1, b1;
33          float result;
34
35          cout << "Enter two integers: ";
36
37          while ( cin >> a1 >> b1 ) {
38
39                  try {
40                          result = funcDE( a1, b1 );
41                          cout << "The result is: " << result << '\n';
42                  }
43                  catch ( DBNE ex ) {
44                          cout << "Exception occurred: " << ex.whatIs() << '\n';
```

```
45                  }
46                  cout << "\nEnter two integers: ";
47          }
48  }
```

```
1   /* Программа 8.6. /H. Schildt/. Замечания по поводу "не перехвата"
2           ошибки - те же. Демонстрирует возможность использования
3           нескольких "выбросов" с несколькими же блоками
4           "перехвата". По аналогии с программой 8.4. имеется локализация
5           try - блока   в функции. (To entercept - перехватить).
6   */
7
8   #include <iostream>
9   using namespace std;
10
11  void funcTest(int X)
12  {
13          try     {
14                  if(X) throw X;
15                  else throw "Value is null.";
16          }
17          catch (int i) {
18                  cout << "Entercept! Exception's number : "
19                          << i << "\n";
20          }
21          catch(char *str) {
22                  cout << "Entercept of string: ";
23                  cout << str << '\n';
24          }
25  }
26
27  void main()
28  {
29          cout << "START\n";
30
31          funcTest(1);
32          funcTest(2);
33          funcTest(0);
34          funcTest(3);
35
36          cout << "END\n";
37  }
```

```
1   /* Программа 8.7. /H. Schildt/. Замечания те же. Демонстрирует catch (...) -
2      перехватить всё остальное. (To entercept - перехватить).
3   */
4
5   #include <iostream>
6   using namespace std;
7
8   void funcTest(int X)
9   {
10          try     {
11                  if(X==0) throw X;
12                  if(X==1) throw 'a';
13                  if(X==2) throw 322.17;
14
15          }
16          catch (int i) {
17                  cout << "Entercept " << i << "\n";
18          }
19          catch(...) {
20                  cout << "Entercept - entercept!\n";
21          }
22  }
23
24  void main()
25  {
26          cout << "START\n";
27
28          funcTest(0);
29          funcTest(1);
30          funcTest(2);
31
32          cout << "END\n";
33  }
```

```
1   /* Программа 8.8. /H. Schildt/. Замечания те же. Демонстрирует повторное
2      генерирование исключения.
3      Перебрасывание исключений (переброшенные исключения).
4   */
5
6   #include <iostream>
7   using namespace std;
8
9   void funcTest()
```

```
{
        try     {
                 throw "Runtime";
        }
        catch(char *) {
                cout << "Entercept of exception in function funcTest().\n";
                throw;
        }
}

void main()
{
        cout << "START\n";

        try     {
                 funcTest();
        }
        catch(char *) {
                cout << "Entercept of exception in function main().\n";
        }

        cout << "END\n";
}
```

```
/* Программа 8.9. Пример перебрасывания исключений.
   Выбрасывается объект исключения класса std::exception.
*/
#include <iostream>
#include <exception>
using namespace std;

void funcTest()
{
        try     {
                cout << "Function funcTest().\n";
                throw exception();
        }
        catch(exception ex) {
                cout << "Exception entercepted in function funcTest().\n";
                throw;
        }
        cout << "This should not out.";
}
```

```
20
21  void main()
22  {
23          try     {
24                  funcTest();
25                  cout << "This also should not out.";
26          }
27          catch(exception ex) {
28                  cout << "Exception entercepted in function main().\n";
29          }
30
31          cout << "Program control continues ater catch in main.\n";
32  }
```

```
1   /* Программа 8.10. /H. Schildt/. Пример обработки исключений,
2      генерируемых оператором new.
3   */
4   #include <iostream>
5   #include <new>
6   using namespace std;
7
8   int main()
9   {
10          int *p, i;
11
12          try     {
13                  p = new int[133];
14          }
15          catch(bad_alloc as) {
16                  cout << "Memory allocation failed for p.\n";
17                  return 1;
18          }
19
20          for(i=0; i<133; i++) p[i] = i;
21
22          for(i=0; i<133; i++) cout << p[i] << " ";
23
24          delete [] p;
25
26          return 0;
27  }
```

```
1   // Программа 8.11. Пример обработки исключений, генерируемых
2   // оператором new.
3
4   #include <iostream>
5   #include <new>
6   using std::cout;
7   using std::bad_alloc;
8
9   int main()
10  {
11          double *p[33];
12
13          try     {
14                  for(int i=0; i<33; i++) {
15                          p[i] = new double[3333333];
16                          cout << "Allocated memory for 3333333 double in p["
17                                  << i << "]\n";
18                  }
19          }
20          catch(bad_alloc as) {
21                  cout << "Exception occured: " << as.what() << '\n';
22          }
23
24          //delete [] p;
25
26          return 0;
27  }
```

```
1   // Программа 8.12. /H. Schildt/. Пример перегрузки (перезагрузки)
2   // операторов new и delete.
3
4   #include <iostream>
5   #include <new>
6   #include <cstdlib>
7   using namespace std;
8
9   class D3 {
10          int x, y , z; // 3d - координаты
11  public:
12          D3() {
13                  x = y = z = 0;
14                  cout << "Making object 0, 0, 0\n";
15          }
```

```
16
17          D3(int i, int j, int k) {
18                  x = i; y = j; z = k;
19                  cout << "Making object " << i << ", ";
20                  cout << " << j << " << ", " << k << '\n';
21          }
22
23          ~D3() { cout << "Destruction object\n"; }
24
25          // Выделение памяти для объекта (синтаксис).
26
27          void *operator new(size_t sz);
28
29          /* Если выделение памяти невозможно - должно
30                  генерироваться исключение типа bad_alloc.
31                  Конструктор вызывается автоматически.
32                  Тип size_t - целочисленный тип без знака,
33                  видимо создан для malloc(). */
34
35          // Выделение памяти для массива объектов (синтаксис).
36
37          void *operator new[](size_t sz);
38          /* Если выделение памяти невозможно - должно
39                  генерироваться исключение типа bad_alloc.
40                  Конструкторы вызываются автоматически */
41
42          void operator delete(void *p);
43          /* Освобождение памяти, адресуемой указателем p. */
44
45          void operator delete[](void *p);
46          /* Освобождение памяти, адресуемой указателем p.
47                  Автоматическивызывается деструктор для
48                  каждого элемента массива. */
49
50          void out();
51  };
52
53  // Перезагрузка №1 оператора new для класса D3.
54  void *D3::operator new(size_t sz)
55  {
56          void *p;
57
58          cout << "Allocated memory for object of class D3.\n";
59
60          p = malloc(sz);
```

```
61          /* Ф-ия С выделения памяти в "куче". Прототип
62                  void *malloc(size_t num_bytes);
63                  возвращает обобщённый указатель - необходимо
64                  привести на требуеиый тип. Успешный вызов ф-ии -
65                  возрат указателя на 1-ый байт памяти "кучи".
66                  Если памяти нет - ф-ия возвращает указатель NULL. */
67
68          // Генерирование исключения в случае
69          // неудачного выделения памяти
70          if(!p) {
71                  bad_alloc ex;
72                  throw ex;
73          }
74          return p;
75  }
76
77  // Перезагрузка №2 оператора new для массива
78  // объектов класса D3.
79  void *D3::operator new[](size_t sz)
80  {
81          void *p;
82
83          cout << "Allocated memory for object's array - type D3.\n";
84
85          p = malloc(sz);
86
87          // Генерирование исключения - неудача
88          if(!p) {
89                  bad_alloc ex;
90                  throw ex;
91          }
92          return p;
93  }
94
95  // Перезагрузка №1 оператора delete для класса D3.
96  void D3::operator delete(void *p)
97  {
98          cout << "Destruction object of class D3.\n";
99          free(p);
100         /* Ф-ия free() С освобождает для с-мы раннее выделенную
101                 память. Является комплементарной ф-ии malloc().
102                 Прототип void *free(void *ptr);
103                 void *ptr - указатель на память, выделенную malloc(). */
104 }
105
```

```
106 // Перезагрузка №2 оператора delete для масива объектов
107 //       класса D3.
108 void D3::operator delete[](void *p)
109 {
110         cout << "Destruction object' array - type D3.\n";
111         free(p);
112 }
113
114 // Ф-ия вывода координат X, Y, Z.
115 void D3::out()
116 {
117         cout << x << ", " << y << ", " << z << "\n";
118 }
119
120 int main()
121 {
122         D3 *p1, *p2;
123
124         try     {
125
126                 /* Выделение памяти для массива объектов - выполнение
127                         варианта №2 перезагр. оператора new. */
128
129                 p1 = new D3[5];
130
131                 /* Выделение памяти для объекта - выполнение
132                         варианта №1 перезагр. оператора new. */
133
134                 p2 = new D3(5, 3, 9);
135         }
136         catch(bad_alloc ex) {
137                 cout << "Memory allocation failed.\n";
138                 return 1;
139         }
140
141         p1[1].out();
142         p2->out();
143
144         delete [] p1; // №2 - удаление массива объектов
145         delete p2; // №1 - удаление объекта
146
147         return 0;
148 }
```

Список использованных источников

1. Введение в C/C++. Учебное пособие по курсу «Прикладное программирование»/Сост. Д.Д. Ветчинин – Иваново, ИГТА, 2010г.–56 с.

2. Введение в VC++. Учебное пособие по курсу «Программная реализация средств дизайна»/Сост. Д.Д. Ветчинин – Иваново, ИГТА, 2012г.–104 с.

3. Г. Шилдт. Базовый курс C++.: . Пер. с англ. – М.: Издательский дом «Вильямс», 2009. – 752 с.: ил.

Содержание

Учебное пособие по курсам «Прикладное программирование в информационных системах», «Программная реализация средств дизайна», «Технологии программирования»
Введение в VC++ MFC
Составители: Дмитрий Давидович Ветчинин
Научный редактор: Николай Анатольевич Коробов
Печатается в авторской редакции

УП № 022307 от 08.04.2013. Подписано в печать 09.04.2013.
Формат 1/16 60*84. Бумага писчая. Плоская печать.
Усл. печ. л. 3,12 Уч.-изд.7,062. Тираж 22 экз. Заказ № 322

РИО ИГТА
ЦОТ кафедры ПМИТ ИГТА
152000, г. Иваново пр. Шереметьевский 21