Программирование

Тема1.1 C++. Введение в классы и объекты

Технологии программирования

- **Процедурное программирование** (с 1957 г., Фортран, Алгол-60 и др);
- **Структурное программирование** (начало 70-х, Паскаль, Си и др);
- **Модульное программирование** (с 1975 г., Модула, Turbo Pascal, Turbo C и др);
- **Объектно - ориентированное программирование** (с середины 80-х, Object Pascal, C++, Java, C# и др.)

- Процедурное программирование заключается в разработке набора функций (алгоритмов) для решения поставленной задачи. Для поддержки этой парадигмы языки программирования предоставляли механизм передачи параметров и получения результатов функций.
- Структурное программирование: – основные положения:

  любую программу можно написать, пользуясь ограниченным набором базовых конструкций :
  - последовательность операторов/блоков,
  - ветвление или выбор ,
  - циклы: с постусловием, цикл с предусловием.

  Каждая из базовых конструкций имеет один вход и один выход.

  “Правильная” структура программы (блока программы, подпрограммы), имеет один вход и один выход. Любую программу можно и нужно писать без использования оператора goto, который очень запутывает структуру программы. Для исключения оператора goto достаточно применять базовые конструкции “цикл”.

  Одновременно с определением набора базовых конструкций была осознана необходимость предварительного анализа данных и конструирования структур данных на начальной стадии разработки программы (В Си для этого есть структуры, объединения и перечисления).
- Модульное программирование: логически связанные между собой подпрограммы и общие данные с которыми они работают (переменные, константы, пользовательские типы данных) размещаются в отдельные модули, части программы, которые компилируются независимо друг от друга.
- Если рассматривать структуру модуля, то модуль состоит из двух частей: интерфейс и реализация. В Си (C++) и интерфейсная часть модуля и реализация размещаются в отдельных файлах. Интерфейсная часть модуля – в заголовочном файле (.h), а реализация модуля в файле реализации (.cpp).

Пример модуля (Си):

```
// Student.h
#ifndef STUDENT_H
#define STUDENT_H

#define MIN_AGE 12 // Константы
#define MAX_AGE 99
typedef struct s{ // Описание новых типов данных
char Name[20];
int Age;
//...
} TStudent;
int InputDataStudent(TStudent* S); // Прототипы функций
void OutputDataStudent(TStudent* S);
int AverageAge();
#endif
```

Пример модуля (Си):

```
// Student.c
#include "Student.h"

// определения функций
int InputDataStudent(TStudent* S){
 //...
}
void OutputDataStudent(TStudent* S){
 //...
}
int AverageAge(){
  //...
}
```

- Объектно-ориентированное программирование развивает идеи модульного программирования по объединению структур данных и алгоритмов их обработки.
- Основными элементами объектно-ориентированной программы в C++ являются *классы и объекты.*
- Класс – это конструкция языка программирования C++, которая объединяет в себе переменные и функции, которая может определять новый тип данных, который можно использовать для создания объектов этого типа.
- Объект в C++ и других объектно - ориентированных языках программирования – это переменная типа класс.
- Переменные, объявленные в классе называют полями данных объекта. Функции, описанные в классе называют методами.

Синтаксис объявления класса в C++:

```
class имя_класса
{ [private:]
   // объявление закрытых полей данных и
   // объявление или определение закрытых методов
  [protected:]
   // объявление защищенных полей данных и
   // объявление или определение
   // защищенных методов
  [public:]
   // объявление открытых полей данных и
   // объявление или определение открытых методов
};
```

- Синтаксис объявления класса в C++: ключевое слово class, за которым следует открывающаяся фигурная скобка, после которой объявляются поля класса и объявляются или определяются методы класса. Объявление класса завершается закрывающейся фигурной скобкой и точкой с запятой.
- Поля класса при объявлении не могут инициализироваться т. е. получать начальные значения.
- Если в объявлении класса метод только объявляется, то он обязательно должен быть определен вне объявления класса. Если метод определяется вне объявления класса, то в заголовке, перед именем метода, указывается имя класса, к которому он относится, и операция доступа к области видимости (::).
- В объявлении класса также могут указываться ключевые слова private, public, protected, которые задают для клиентов класса степень доступа к полям и методам класса.
- Клиентами класса могут быть другие классы и внешние функции (например, main()).
- Поля и методы, описанные в классе после ключевого слова private являются закрытыми, т. е. доступными только для методов этого класса;
- По умолчанию все поля и методы класса считаются закрытыми (private).
- Поля и методы, описанные в классе после ключевого слова protected являются защищенными, т. е. доступными только для методов этого класса и классов-потомков.
- Поля и методы, описанные в классе после ключевого слова public являются открытыми, т. е. доступными для любых клиентов класса;
- Ключевое слово степени доступа (private, public, protected) применяется ко всем расположенным ниже полям и методам до тех пор, пока не встретиться следующее ключевое слово степени доступа. Это позволяет создавать в объявлении класса разделы для открытых, закрытых и защищенных полей и методов.
- Один из принципов объектно-ориентированного программирования, который называется инкапсуляция, требует чтобы поля данных класса объявлялись в секции private. Методы класса, необходимые клиентам класса, объявляют в секции public.

Пример 1:

```
// cat.h
// объявление класса Cat
 class Cat
 { private: int age; // возраст
                char name[20]; // кличка
   public:
  // определения методов getAge и setAge
     int getAge(){ return age;}
     void setAge(int Age){ age = Age;}
   void Meow(); // объявление метода Meow()
 };
```

```
// cat.cpp
#include "cat.h"
#include <iostream>
using namespace std;

// определение метода Meow() класса Cat
  void Cat::Meow()
    {   cout << name << ": ";
       for (int i = 1; i <= age; i++)
          cout << "Мяу ";
       cout << "\n" <<  "\n" ;
    }
```

- Пример 1: в классе Cat поля данных age и name объявлены в секции private. Клиенты класса могут получить и изменить значение поля age с помощью методов getAge() и setAge(). К значению поля name доступ для клиентов класса закрыт.

- Объявление класса (Например, объявление класса Cat) обычно размещается в заголовочном файле модуля (cat.h) и называется интерфейсом класса, поскольку оно сообщает пользователю, как взаимодействовать с классом.
- Определения методов класса (Например, определения методов класса Cat) обычно размещается в файле реализации модуля (cat.cpp) и называется реализацией класса.
- Заголовочный файл модуля подключают в файл реализации модуля с помощью директивы #include (Например, #include "cat.h“). См. пример1.
- Файл реализации класса (Например, файл cat.cpp) всегда нужно добавлять в проект программы, которая является клиентом класса. В программу-клиент необходимо включать интерфейс класса (Например, #include "cat.h“).) .

Конструкторы и деструкторы

- **Конструктор** – метод класса, который автоматически вызывается после выделения памяти под объект (после создания объекта).
- **Деструктор** – метод класса, который автоматически вызывается перед освобождением памяти из под объекта (перед уничтожением объекта).

Виды конструкторов:

- Конструкторы по умолчанию (без параметров);
- Конструкторы с параметрами;
- Конструкторы копирования.

- Конструктор используется для инициализации полей объекта, а также для выделения памяти для динамических полей объекта (т. е. полей, память под которые необходимо выделять на этапе выполнения программы).
- Основные свойства конструктора: имеет тоже имя, что и класс; не может возвращать значение.
- Конструкторы можно перегружать для различных вариантов инициализации объектов класса.
- Замечание. В C++ методы класса и внешние функции могут быть перегружены, т. е. иметь одинаковые имена, но разные списки параметров.
- Если в классе отсутствует конструктор, то компилятором автоматически создается конструктор по умолчанию (конструктор без параметров), который не выполняет инициализацию полей объекта класса.
- Если класс имеет конструктор с параметрами, то конструктор без параметров (конструктор по умолчанию) автоматически не создается, поэтому, если он необходим, то его необходимо создавать явно (т. е. перегрузить конструктор с параметрами).
- Деструктор используется для выполнения тех действий, которые необходимо выполнить перед уничтожением объекта, например, освобождает память, выделенную конструктором для динамических полей. Имя деструктора совпадает с именем класса и начинается с символа тильда (~). Деструктор не имеет параметров и не может возвращать значение. Если деструктор не объявлен явно, он, как и конструктор, создается компилятором как пустой метод.
- Класс может иметь только один деструктор, перегрузка деструкторов запрещена.
- Замечание: В примере 2 (на след. слайде) модификатор const в параметрах конструктора защищает эти параметры от случайных (ненамеренных) изменений. В случае изменения этих параметров компилятор выдает сообщение.

Пример 2:

```
// cat.h
// объявление класса
 class Cat
 { private: int age; // возраст
            char name[20]; // кличка
   public:
    Cat(); //конструктор_1
    Cat(const char Name[], int Age); //конструктор_2
    ~Cat(){}; //деструктор
    int getAge(); //возвращает возраст
    void setAge(int Age);//устанавливает возраст
    void Meow();
 };
```

```
// cat.cpp
 #include "cat.h"
#include <iostream>
#include <cstring>
using namespace std;

// определение методов класса
  Cat::Cat(){ strcpy(name, "Cat"); age = 1;}
  Cat::Cat(const char Name[], int Age)
    {strcpy(name, Name); age = Age;}

  int Cat::getAge(){ return age;}
  void Cat::setAge(int Age){ age = Age;}

  void Cat::Meow()
    {  cout << name << ": ";
       for (int i = 1; i <= age; i++)
         cout << "Мяу ";
       cout << "\n" <<  "\n" ;
    }
```

- **Варианты создания статического объекта:**

```
- имя_класса   имя_объекта;
- имя_класса   имя_объекта(список параметров конструктору);
```

- **Варианты создания динамического объекта:**

```
- имя_класса  * имя_ук = new имя_класса;
- имя_класса  * имя_ук = new имя_класса (список параметров конструктору);
```

- **Доступ к полям и методам, объявленным в секции public:**

```
имя_объекта.имя_поля
имя_объекта.имя_метода()
указатель_на_объект -> имя_поля
указатель_на_объект -> имя_метода()
```

Указатель this

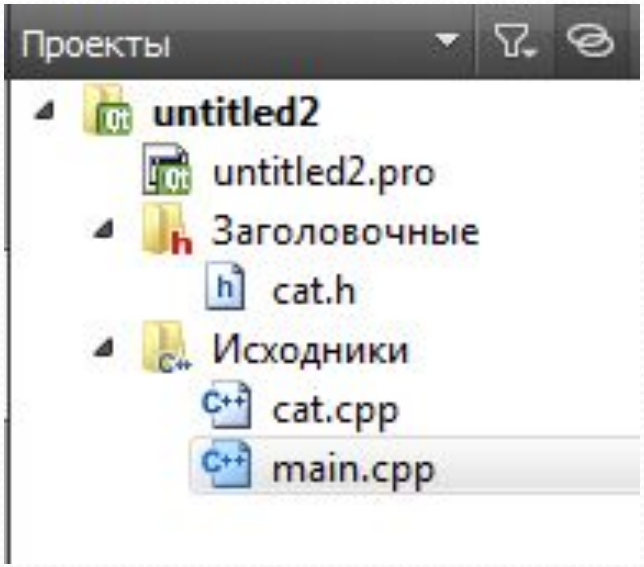
```
int Cat::getAge(){ return this->age;}
void Cat::setAge(int age){ this->age = age;}
```

- На основе класса в программе создаются экземпляры этого класса (объекты).
- В зависимости от способа создания объекта (выделения памяти под объект) объекты могут быть статическими и динамическими.
- Под статические объекты память отводит компилятор в стеке или глобальной области памяти, а под динамические объекты – программист с использованием операции динамического выделения памяти (new). Память по динамические объекты отводится в динамической области памяти (куче, heap).
- Если поля и методы объявлены в секции public, то к ним разрешен *доступ* за пределами класса любым функциям программы с использованием имени объекта и операций точка или стрелка (точка – используется для статических объектов, а стрелка – для динамических объектов).
- Для доступа за пределами класса к значениям закрытых полей данных (объявленных в секции private) используют public методы доступа:
  - методы записи (изменения значения поля). Имена этих методов рекомендуют начинать с set;
  - методы чтения (получения значения поля). Имена этих методов рекомендуют начинать с get;
- При создании объекта класса, пространство в памяти отводится только под поля объекта. Для методов память при создании объекта не резервируется. Каждый метод класса существует в памяти в единственном экземпляре и может работать с полями данных каждого объекта класса. Для этого каждый метод класса, кроме статических методов, (методов объявленных в классе с ключевым словом static) имеет скрытый параметр – указатель this, в который передается адрес объекта, который вызывает метод. Указатель this неявно применяется для доступа к полям объекта из методов объекта. Но при желании можно явно применить указатель this для доступа к полям объекта из методов объекта, особенно если параметры метода совпадают по именам с полями объекта.

Продолжение примера 2:

```
// main.cpp
#include "cat.h"
int main()
{ Cat Cat1; //статический объект, вызывается
            // конструктор без параметров
  Cat Cat2("Murzik",3); //статический объект,
            // вызывается конструктор с параметрами
  Cat * Cat3 = new Cat; //динамический объект
  Cat * Cat4 = new Cat("Pushok",4); //динамический объект
  Cat1.Meow(); Cat2.Meow();
  Cat3->Meow(); Cat4->Meow();
  // Cat1.age = 5; Ошибка!!!
  Cat2.setAge(5);//Правильно!!!
  Cat4->setAge(6);
  cout << "\n" <<  "\n" ;
  Cat2.Meow();  Cat4->Meow();
  delete Cat3; delete Cat4;
}
```

Состав проекта и пример выполнения программы примера 2:




```
Cat: Мяу

Murzik: Мяу Мяу Мяу

Cat: Мяу

Pushok: Мяу Мяу Мяу Мяу


Murzik: Мяу Мяу Мяу Мяу Мяу

Pushok: Мяу Мяу Мяу Мяу Мяу Мяу
```

Конструктор копирования

- **Варианты вызова конструктора копирования:**

```
AnyClass ob2 = ob1;//ob1 явно инициализирует ob2
Func1(ob1); //ob1 передается в качестве параметра
ob2 = Func2(ob1); //ob2 получает значение возвра-
                  // щаемого объекта
```

- **Синтаксис конструктора копирования:**

```
имя_класса (const имя_класса &obj)
{ тело_конструктора }
```

- obj – ссылка на копируемый объект

- Конструктор копирования вызывается в трех случаях:
  - когда в операторе объявления один объект используется для инициализации другого;
  - при передаче объекта в функцию по значению ;
  - при создании временного объекта для возврата объекта из функции.
- В конструктор копирования передается ссылка на копируемый объект.
- Замечание 1. Ссылки в C++ фактически является скрытыми указателями, синтаксически работают как псевдонимы переменных.

  Пр.: int n = 5; int * p = &n; // в указателе p лежит адрес n

  *p = 7; // по адресу, находящемуся в указателе p размещаем значение 7, в n лежит 7

  int &r = n; // в ссылке r лежит адрес n

  cout << r; // по адресу, находящемуся в ссылке r получаем значение – это 7 и выводим

  r = 9; // в n лежит 9

  В отличие от указателей ссылки всегда привязаны к конкретному объекту:

  int *p; // компилируется;

  int &r; // ошибка компиляции!;

  int &r = *p; // а так пойдёт (p – не инициализирован!).

  После инициализации ссылку нельзя привязать к другому объекту.
- Замечание 2. Передавая функции const-ссылку на переменную и копия переменной не создается, и исходная переменная не будет подвергнута никаким случайным изменениям (компилятор выдает сообщение об ошибке). Const-ссылки очень часто используют при передаче объекта как параметра функции. Объекты в функцию можно передавать по значению т. е. создавать копии объектов. В результате создания копий объектов (из-за их внушительных размеров) могут возникнуть нежелательные побочные эффекты. Поэтому для передачи в функцию объектов, которые в ней не должны изменяться, лучше использовать const-ссылки.

- Конструктор копирования, заданный по умолчанию (неявно), выполняет поэлементное копирование значений нестатических полей класса. При этом выполняется копирование по значению (побитовое копирование). В результате такого копирования могут возникнуть проблемы, например, при побитовом копировании объекта, содержащего в качестве поля динамический массив (будут скопированы не сами элементы массива, а его адрес).
- Для решения подобных проблем применяют явно заданные конструкторы копирования.
- Следует помнить, что передача объекта в функцию по ссылке предотвращает вызов конструктора копирования. Это позволяет сберечь время, необходимое для его вызова и сэкономить память, используемую для хранения нового объекта.
- Конструктор копирования используется только при инициализации. Конструктор копирования никогда не участвует в процессе присваивания и никак не влияет на него.

Пример 3:

```
// cat.h

// объявление класса
 class Cat
 { private: int age;     // возраст
           char* name; // кличка
   public:
    Cat(); Cat(const char* Name, int Age);
    ~Cat(){delete []name; }
    Cat(const Cat &c);
    int getAge();          //возвращает возраст
    void setAge(int Age);//устанавливает возраст
    char* getName(); //возвращает кличку
  void setName(const char* Name);//устанавливает кличку
    void Meow();
 };
```

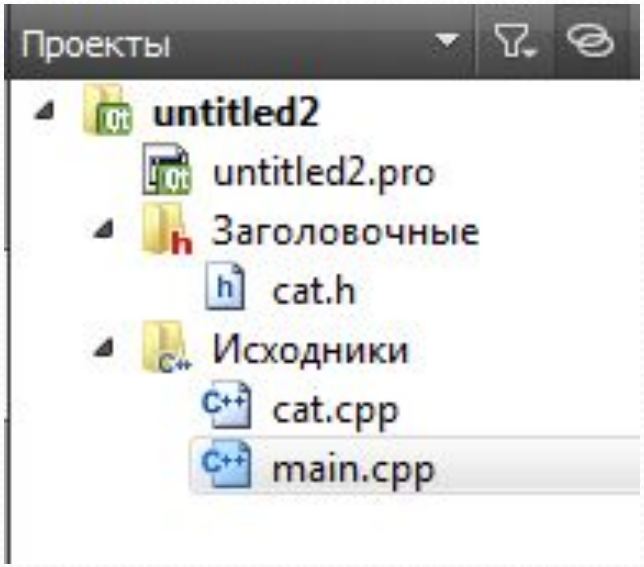
```
// cat.cpp
#include "cat.h"
#include <iostream>
#include <cstring>
using namespace std;
// определение методов
Cat::Cat()
 { name=new char[2];//выделяем память
   strcpy(name,"");      //копируем имя
   age = 1;
 }
Cat::Cat( const char* Name, int Age)
 { int leng=strlen(Name); //выч.длины имени
   name=new char [leng+1];//выделяем память
   strcpy(name,Name);      //копируем имя
   age = Age;
 }
```

```
char* Cat::getName() {return name;}

 void Cat::setName(const char* Name)
  {int leng=strlen(Name); //выч.длины имени
   name=new char [leng+1];//выделяем память
   strcpy(name,Name);
  }

int Cat::getAge(){ return age;}

void Cat::setAge(int Age){ age = Age;}
```

```
//определение конструктора копирования
Cat::Cat(const Cat &c)
 {int leng=strlen(c.name);//выч-е длины имени
  name=new char[leng+1]; //выд-е памяти для копии
  strcpy(name,c.name);      //копируем имя
  age=c.age;                //копируем возраст
 }

void Cat::Meow()
 {  cout << "\n"<< name << ":";
    for (int i = 1; i <= age; i++)
     cout << "Мяу ";
 }
```

```
// main.cpp
#include "cat.h"
Cat NoName(Cat& cat)
{ Cat temp(cat);
  temp.setName("NoName");
  return temp;}
void view(Cat cat) {cat.Meow();}
int main()
{ Cat Cat1("Пушок",2); Cat1.Meow();
  Cat Cat2 = Cat1; //вызов констр-ра копирования
  Cat2.setAge(4);
  view(Cat2); //вызов констр-ра копирования
  Cat Cat3 = NoName(Cat2);//вызов констр. копирования
   Cat3.Meow();
   Cat3.setName("Мурзик");
   Cat3.Meow();  }
```

Состав проекта и пример выполнения программы примера 3:

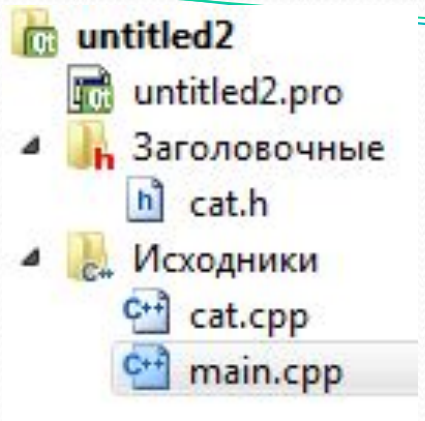



```
Пушок:Мяу Мяу
Пушок:Мяу Мяу Мяу Мяу
NoName:Мяу Мяу Мяу Мяу
Мурзик:Мяу Мяу Мяу Мяу
```

Способы создания массива объектов

```
1. Cat Family1[5]; //статический массив объектов
                   // класса Cat -  для каждого
  //объекта вызывается конструктор без параметров
2. Cat * Family2;
   Family2 = new Cat[5]; //динамический массив
         // объектов класса Cat - для каждого объекта
         // вызывается конструктор без параметров
3. Cat * Family3[5]; //массив указателей на объекты
                     // класса Cat

  for (int i=0; i < 5; i++)
      Family3[i] = new Cat;
```

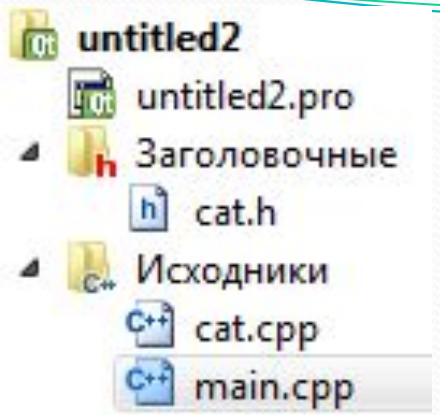
Пример 4.




```
Возраст котов (в годах):
Кот N 1 - 1
Кот N 2 - 3
Кот N 3 - 5
Кот N 4 - 7
Кот N 5 - 9
```

```
// main.cpp
#include "cat.h"
#include <iostream>
using namespace std;
int main()
{ Cat Family1[5]; // статический массив объектов
  for (int i=0; i < 5; i++)
    Family1[i].setAge(2*i+1);
 cout << " Возраст котов (в годах):";
  for (int i=0; i < 5; i++)
    cout << "\n Кот N "<< i+1 << " - " <<
  Family1[i].getAge();
}
```

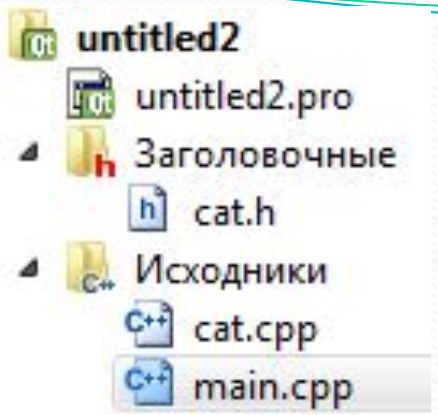
Пример 5.




```
Возраст котов (в годах):
Кот N 1 - 2
Кот N 2 - 4
Кот N 3 - 6
Кот N 4 - 8
Кот N 5 - 10
```

```
// main.cpp
#include "cat.h"
#include <iostream>
using namespace std;
int main()
{Cat* Family2 = new Cat[5]; // динамический массив
  объектов
   for (int i=0; i < 5; i++)
     Family2[i].setAge(2*i+2);
 cout << " Возраст котов (в годах):";
   for (int i=0; i < 5; i++)
     cout << "\n Кот N "<< i+1 << " - " <<
  Family2[i].getAge();
   delete []Family2;
}
```

Пример 6.




```
Возраст котов (в годах):
Кот N 1 - 3
Кот N 2 - 5
Кот N 3 - 7
Кот N 4 - 9
Кот N 5 - 11
```

```
// main.cpp
#include "cat.h"
#include <iostream>
using namespace std;
int main()
{ Cat* Family3[5]; // массив указателей на объекты
  for (int i=0; i < 5; i++)
   { Family3[i] = new Cat;
     Family3[i]->setAge(2*i+3);}
  cout << " Возраст котов (в годах):";
  for (int i=0; i < 5; i++)
   cout << "\n Кот N "<< i+1 << " - " <<
              Family3[i]->getAge();
  for (int i=0; i < 5; i++)
   delete Family3[i];
}
```

Примеры использования инициализаторов для инициализации объектов

```
1. Cat Family1[3] = { Cat("Пушок1", 3),
                      Cat("Мурзик", 5),
                      Cat("Пушок2", 2) };

2. Cat* Family3[3] = { new Cat("Пушок1", 3),
                       new Cat("Мурзик", 5),
                       new Cat("Пушок2", 2) };
```

Контрольные вопросы

1. Понятие модуля, структура модуля в С(С++). Пример модуля.
2. Понятие класса в С++. Синтаксис описания класса в С++. Понятие интерфейса и реализации класса. Примеры.
3. Понятие, назначение и основные свойства конструкторов и деструкторов. Виды конструкторов, примеры их использования.
4. Понятие объекта в С++. Варианты создания объекта в С++: синтаксис и примеры. Как организуется доступ к полям и методам объекта, объявленным в секциях private и public?
5. Назовите способы создания массива объектов, приведите примеры.