# ЕН.Ф.02 – Информатика и программирование

## Лекция 5. Функции. Модульный стиль программирования

Конова Елена Александровна
E_Konova@mail.ru

# План

- Функции. Основные определения.
- Категории функций.
- Описание и объявление функции.
- Обращение к функции:
  - оператор-выражение;
  - оператор-функция.
- Механизм обращения к функции.
- Передача параметров в функцию:
  - Параметр, передаваемый по значению;
  - Параметр, передаваемый по адресу .
- Перегруженные функции.
- Область действия и время жизни объектов .
- Локальные и глобальные объекты.
- Принцип локализации имен.

# Функции

Определение. Поток управления.

Определение. Функция – самостоятельный именованный алгоритм решения некоторой задачи.

Функция:

- может использоваться многократно,
- может быть подключена к любому проекту,
- безошибочно решает задачу при любых наборах данных,
- может входить в библиотеку.

Пример функции в проекте Functions.

# О структуре программного кода

Все функции в C++ описываются на одном уровне.

С функции **`main`** начинается выполнение кода.

**`main`** способна вызвать другие функции, а другие функции способны вызывать другие функции.

Процедура вызова означает передачу управления из одной программной единицы другой.

Механизм взаимодействия программных единиц друг с другом, это разделение передачи управления и передачи данных.

Принцип сокрытия данных внутри функции.

# Категории функций

1. Библиотечные функции.
2. Функции, определенные пользователем.

# 1. Библиотечные функции

1. Объявления библиотечных функций содержатся в заголовочных файлах с именами <имя.h>. Чтобы объявление стало доступно программе, необходимо включить в текст программы директиву, например:

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

Текст заголовочного файла, в котором есть объявление функции, включается в текст программы на этапе препроцессорной обработки.

В некотором роде, это аналог объявлению переменных.

2. Описания библиотечных функций содержатся в библиотечных файлах. Они хранятся в компилированном виде (файлы с расширением **obj**), присоединяются к коду разработчика на этапе сборки.

# 2. Функции пользователя

Собственные функции пользователя описываются:

1) в теле программы, тогда описание функции записывается внутри файла программы;

2) в заголовочных файлах, тогда это файл с именем "имя.h", который содержит тексты описания функций.

Смысл – заголовочный файл легко подключается к любой программе, использующей эти функции.

Например,

```
#include "имя.h"
```

Препроцессор ищет необходимый включаемый файл сначала в текущем каталоге, потом в специальном каталоге **include**.

# Описание и объявление функции

Функция должна быть описана перед употреблением

Синтаксис описания функции:

Тип_функции Имя_Функции (Список формальных параметров)

{

// Описание локальных данных.

    Тело функции − блок.    // Описание алгоритма.

}

# Пояснения к синтаксису

Функция имеет тип:

1) если функция возвращает значение базового типа, то ее тип – один из базовых типов (в теле функции оператор **`return`** возвращает в вызывающую функцию найденное значение);
2) если функция не возвращает значения, ее тип **`void`** – такими бывают, например, интерфейсные функции: в теле функции **`return`** может отсутствовать или быть записан как

```
return ;        // Значение не возвращается
```

3) если функция возвращает значение сложного типа (массив, объект, запись и прочие), то ее типом должен быть указатель (адрес объекта);
4) Если тип функции не указан, то тип будет **`int`**.

# Пояснения к синтаксису

Имя функции должно быть осмысленным и уникальным в проекте.

Не рекомендуется переопределять имена библиотечных функций, в этом случае вступает в действие механизм перегрузки функций: одно имя сопоставлено разным алгоритмам, например:

```
float sin (float x)
   {
        // Новое описание.
   }
```

Не рекомендуется начинать имена со знака подчеркивания, так начинаются макросы, например:

```
_max (int a, int b)
```

# Пояснения к синтаксису

Параметры функции

Параметры функции в описании – формальные, то есть не имеют реальных значений, а только описывают типы, количество и порядок следования данных, передаваемых в функцию. Представлены именем.

Термин – сигнатура параметров.

Параметры функции в обращении – фактические, это те реальные значения, с которыми функция отрабатывает очередной вызов.

Фактическими параметрами могут быть константы, переменные, выражения, указатели.

# Тело функции

В теле функции решается задача обработки данных. Данные передаются функции через список параметров.

1. Описания локальных переменных. Их область действия – только тело функции.
2. Описание алгоритма содержит детали реализации.
3. Возврат в точку вызова обеспечивает оператор **return**, который может иметь две формы:

```
return;              // для функции типа  void.
return Выражение;
 // Тип выражения совпадает с типом функции.
```

# Обращение к функции

Обращение к функции, это передача управления в тело функции, а также передача фактических данных, с которыми функция отрабатывает вызов.

Синтаксис:

Имя_Функции (Список фактических параметров)

По схеме передачи данных из функции различают:

- оператор-выражение.
- оператор-функция.

# Оператор-выражение

Если функция возвращает одно значение, то в точку вызова передается одно значение, не обязательно базового типа.

Это значение можно:

а) присвоить, например:

```
S = Square(2,3,4);
```

б) использовать в выражении, например:

```
S = 2*Square(2,3,4);
```

в) напечатать, например:

```
printf("Square is %6.2f\n",Square(2,3,4));
```

# Оператор-функция

Если функция не возвращает значения, то обращение к ней выглядит как обычный оператор программы:

```
printf("%d %d %d\n",a,b,c);
```

Пример.

# Механизм обращения к функции

Обращение к функции, это:

передача управления алгоритму функции;

передача фактических данных для решения задачи.

Пример обращения:

```
Square(2,3,4);
```

# Механизм обращения к функции

1. Управление передается в функцию по оператору обращения.
2. Выделяется память для параметров функции, вычисляются их значения и копируются в локальную память.
3. Создаются локальные переменные функции, которые живут только в теле функции.
4. Выполняется алгоритм функции, который использует внешние и локальные данные.
5. По **`return`** управление передается в точку вызова, в вызывающую программу передается возвращаемое значение.
6. Локальные переменные умирают, память высвобождается.

Процессы видны в отладчике.

# Описание и объявление функции

**Описание** функции предоставляет всю информацию о ней:

- внешнюю – тип, имя, сигнатура параметров,
- внутреннюю – тонкости реализации алгоритма.

**Объявление** функции предоставляет только внешнюю интерфейсную часть, для которой важно только умение правильно использовать функцию, то есть обратиться к ней.

Механизм: описание функции должно быть известно компилятору, который должен обработать вызов функции (сопоставить обращение реальному алгоритму).

Поэтому место объявления или описания функции в программе – до первого обращения к ней.

# Варианты объявления функции

1. Описание функции фактически расположено перед текстом вызывающей программы, тогда описание и объявление – одно и то же, появится перед первым вызовом функции.
2. Описание функции фактически расположено после текста вызывающей программы, и вызов функции появится раньше того, как описание известно. Необходимо объявление функции.

Объявление функции (прототип) – это ее заголовок, за которым стоит знак «;».

Прототип может быть записан в начале программы, если функция глобальна, или в теле **`main`**, если функция локализована в **`main`**, или в теле любой функции.

# Варианты объявления функции

3. Наилучший вариант описания функций пользователя, это заголовочные файлы, которые содержат описания множества функций, и легко подключаются к любому проекту, где необходимо их использование.

Библиотеки пользователя могут группировать функции по логике использования, например:

- функции работы с массивами;
- геометрические функции;
- интерфейсные функции;
- и так далее.

# Передача параметров в функцию

Глобально в C++ существуют два способа передачи параметров в функцию.

1. По значению. Данные, передаваемые в функцию, не изменяются ею, и могут быть только входными аргументами.
2. По ссылке (по адресу). Данные, передаваемые в функцию, могут быть ею изменены, и могут быть как входными данными, так и результатом работы функции.

# Параметр, передаваемый по значению

Синтаксис:

Тип Имя_параметра

Механизм: создается локальная копия параметра в теле функции.

Выводы:

1) значения параметров при обращении к функции не могут быть изменены;
2) это мощное средство защиты внешних данных от случайного их изменения функцией;
3) фактические параметры могут быть константами, переменными, выражениями.

# Параметр, передаваемый по адресу

Синтаксис:

Тип & Имя_параметра

// &-признак адресной операции.

Механизм: функция и вызывающая программа работают с адресом объекта в памяти (с одной и той же областью данных, выделенной объекту в вызывающей программе).

Выводы:

1) параметр является единым объектом для функции и для вызывающей программы;
2) функция может изменить значения переданных ей параметров;
3) фактический параметр, передаваемый по ссылке, может быть только адресуемым данным, это переменная.

# Перегруженные функции

Принцип: одно имя может реализовать разные алгоритмы. Значит, можно описать несколько функций с одним и тем же именем.

Они должны отличаться друг от друга:

а) количеством параметров;

б) типом параметров.

Использование одного имени в разных целях называется перегрузкой.

Примеры перегрузки.

# Варианты перегрузки

Возможна перегрузка :

1) по числу параметров,

2) по типу параметров.

Таким образом, перегруженные функции различаются по сигнатуре параметров.

Механизм работы перегруженных функций основан на обработке вызовов, в которых показана сигнатура фактических параметров.

Их типы или их количество сообщают компилятору, какой именно функции сопоставить данный вызов.

Пример. Функция печати.

# Область действия и время жизни объектов

Определение: Объект, это сущность, обладающая некоторыми атрибутами (свойствами) и методами для проверки и изменения атрибутов объекта.

Программные объекты представлены своим именем:

переменная,

именованная константа,

функция.

Каждый объект должен быть объявлен. Роль объявления – задать имя объекта и атрибуты.

Механизм объявления:

1) выделить память,

2) может быть, определить значение.

# Определения

Область действия – это область программного кода, в которой объект известен (то есть действует его объявление).

Время жизни – понятие, связанное с областью действия, это период времени в процессе выполнения программы, когда объект фактически занимает память.

# Локальные и глобальные объекты

Локальные (внутренние) объекты объявлены внутри тела блока, в том числе блока тела функции, например:

```
{
  int a;
  …
}
```

Область действия – блок, в котором описан объект, от точки описания до конца блока.

Время жизни локального объекта – только время выполнения блока: выделяется память при входе в блок, при выходе память освобождается.

Пример в отладчике.

# Локальные и глобальные объекты

Глобальные (внешние) объекты объявлены вне тела всех функций на внешнем уровне.

```
int x;
void main(void)
{
}
```

Область действия – от точки объявления до конца файла с кодом программы.

Время жизни – время выполнения программы.

Замечание. Запрещается использовать глобальные данные (кроме функций и констант).

# Параметры функций

Параметры функций по механизму действия также разделяются на локальные и глобальные, а именно:

- Параметр, передаваемый по значению – локальный, так как создает копию данного в теле функции.

Порождены функцией, живут только в теле функции.

- Параметр, передаваемый по ссылке – глобальный, так как это адрес объекта  в памяти.

Порождены вне функции, область определена вне функции, как и время жизни.

# Принцип локализации имен

Вопрос. Что произойдет, если имена локального и глобального данного одинаковы?

Ответ. Действует принцип локализации имен (пространство имен, приоритет имен, сокрытие имен).

Механизм: на время действия локальной переменной, глобальная переменная с тем же именем временно прекращает свое существование и возобновляет свое значение, как только локальная переменная умерла.

Пример в отладчике.

# Классы памяти

Классифицируют переменные по схеме выделения памяти, меняют механизм глобализации имен.

Приписываются перед объявлением типа переменной, например,

```
static int n;
```

Любой объект приписан к определенному классу памяти, по умолчанию **auto**.

**auto** автоматически определены механизмы локализации.

**static** средство глобализации объекта из тела функции.

**register** данное размещается в регистрах процессора, является рекомендательным.

# Статические объекты

Синтаксис:

**`static`** Тип Имя;

Как глобальное имя имеет мало смысла.

В теле функции объект **`static`**:

получает память один раз при старте программы,
обнуляется.

При повторных входах в функцию значение сохраняется.

Пример.

# Модульный стиль программирования

Прикладная программа на любом языке программирования разрабатывается в IDE – интегрированной среде разработчика.

Программа содержит несколько файлов исходного кода, и каждый файл – это множество функций обработки данных.

Функции объединяются в модули, модули входят в состав проекта, проект входит в состав решения (Solution).

Язык также содержит библиотеки функций, в которых в объектном виде хранятся коды функций обработки данных.

# Определения

**Модуль** – отдельный файл, в котором группируются функции и связанные с ними данные. Решает обособленную задачу обработки данных.

**Достоинства** модульного стиля программирования.

- Алгоритмы отделены от данных. Данные, как правило, имеют какую-то структуру. Алгоритмы обработки данных определяются составом и структурой данных.
- Достигается высокая степень абстрагирования проекта использованием функциональной декомпозиции.

# Механизмы

Модули отлаживаются отдельно друг от друга. Чем больше они независимы друг от друга, тем легче процесс отладки.

Функции, входящие в состав каждого модуля, представляют его интерфейс. Для использования модуля достаточно знать только его интерфейс, не вдаваясь в подробности реализации.

# Функциональная декомпозиция

Функциональная декомпозиция, это метод разработки программ, при котором задача разбивается на ряд легко решаемых подзадач, решения которых в совокупном виде дают решение исходной задачи в целом.

Проектирование приложения строится от абстрактного описания основной задачи (высший уровень абстракции). Основная задача может быть разбита на ряд более простых подзадач (второй уровень абстракции), каждая из которых, в свою очередь, на ряд еще более простых.

Процесс детализации заканчивается, когда очередная подзадача не может быть больше разбита на более простые составляющие, или когда решение очередной задачи становится очевидным.

# Функциональная декомпозиция

В процессе детализации создается иерархические дерево решения, где каждый уровень представляет собой решение более детализированной задачи, чем предшествующий уровень.

Каждый блок представляет собой программный модуль. Каждый модуль, это законченный алгоритм решения некоторой конкретной задачи.

Процесс кодирования выполняется снизу вверх, от написания и полной отладки кода небольших подзадач с их последующей сборкой на верхнем уровне, при этом каждый модуль безошибочно решает одну задачу.

Объем задачи нижнего уровня достаточно небольшой, это одна или две страницы кода.