Подпрограммы. Организация функций

Что такое подпрограмма (функция) и для чего она нужна?

Во многих языках программирования, в том числе и в C++, есть замечательная особенность: можно разделить код большой программы на подпрограммы – функции. Подпрограмма – это такой фрагмент программного кода, которому программист дает уникальное имя. В дальнейшей разработке программист может обращаться по этому имени к подпрограмме, чтобы она выполнила ту задачу, решение которой в ней было реализовано. При этом каждую из этих функций можно расположить в отдельном файле.

Для имени каждой функции должно быть подобрано такое название, по которому сразу можно определить, какая именно задача была реализована в данной конкретной функции. Например, если функция должна суммировать числа, то эту функцию можно назвать «Sum()»; если функция должна подсчитывать среднюю стоимость, то – «AverageCost()» или «AvgCost()». При дальнейшем использовании этой функции нам уже не обязательно помнить, как именно функция реализует свою задачу, а просто достаточно понимать, для чего нужна эта функция. Поэтому выбор имени для функции – важная часть разработки программы.

Итак, возможность разбивать программу на несколько разных функций имеет неоспоримые преимущества:

- значительно снижается сложность написания больших программ: единожды написав функцию, можно многократно ее использовать, причем даже в других программах;
- упрощается чтение кода таких программ: ведь количество строк в функции main() существенно сокращается;
- появляется возможность в простейшей форме реализовать механизм инкапсуляции для сокрытия внутренней реализации фрагмента кода с помощью функции.

Структура функции

Общая структура функции выглядит следующим образом:

```
<тип возвращаемого значения> ИмяФункции
(<аргументы функции>)
{
    //здесь реализуются задачи функции
    return <какое-то полученное значение>
}
```

Аргументы функции – это такие значения, которые функция получает при вызове. Например, если функция должна посчитать сумму двух чисел, то в качестве аргументов можно передать в функцию именно эти два числа.

Если функция не принимает никаких аргументов, то в скобках можно написать ключевое слово void или вообще оставить скобки пустыми, например:

```
int RandomNumber ()
{
    srand(time(0));
    guessedNumber = 1 + rand() % 100;
}
```

Возвращаемое значение – это то, ради чего была вызвана функция. Т.е. этот тот результат, который получила функция в результате своей работы. Функция возвращает этот результат в ту часть программного кода, в котором производился вызов данной функции. Например, если у нас есть функция

```
double AvgCost (double firstPrice, secondPrice, thirdPrice) {}
```

и мы во время ее вызова передадим ей в качестве параметров значения 10.5, 14.5 и 17, то она найдет среднее арифметическое этих чисел, а затем вернет значение 14.0 в вызывающую функцию.

Функция может быть настолько “самодостаточной”, что в ней могут вообще отсутствовать и аргументы, и возвращаемое значение. Также могут отсутствовать только аргументы либо только возвращаемое значение.

Например, если функция сама заботится о том, чтобы вывести результат своих вычислений на экран монитора, то такая функция может не иметь возвращаемого значения. В таком случае вместо типа данных перед именем функции пишется ключевое слово void:

```
void DisplaySum (double firstValue, double secondValue)
{
    cout << firstValue + secondValue << endl;
}
```

Тело функции

У каждой функции должна быть некая содержательная часть, ограниченная фигурными скобками – тело функции, т.е. такая область, где реализуется та самая задача, для которой мы написали данную функцию.

При написании программы на языке C++ все фигурные скобки в редакторе исходного кода любой среды разработки должны находиться строго друг над другом. Такое расположение помогает разработчику визуально разделять код на блоки. Это же относится и к функциям: тело функции должно располагаться внутри фигурных скобок таким образом, чтобы каждая скобка целиком занимала целую строку кода.

Где можно располагать функции?

1 способ:

```
1   #include <iostream>
2   using namespace std;
3
4   int myFunction()
5   {
6       int x = 5;
7       int y = x + 25;
8       return y;
9   }
10
11  int main()
12  {
13      int itog = myFunction();
14
15      system("pause");
16  }
```

Функцию, которую написал сам программист, можно расположить в том же файле, где находится функция main(). В этом случае описание функции должно располагаться перед функцией main(). Если размеры программы и функции небольшие, то этот вариант расположения является приемлемым.

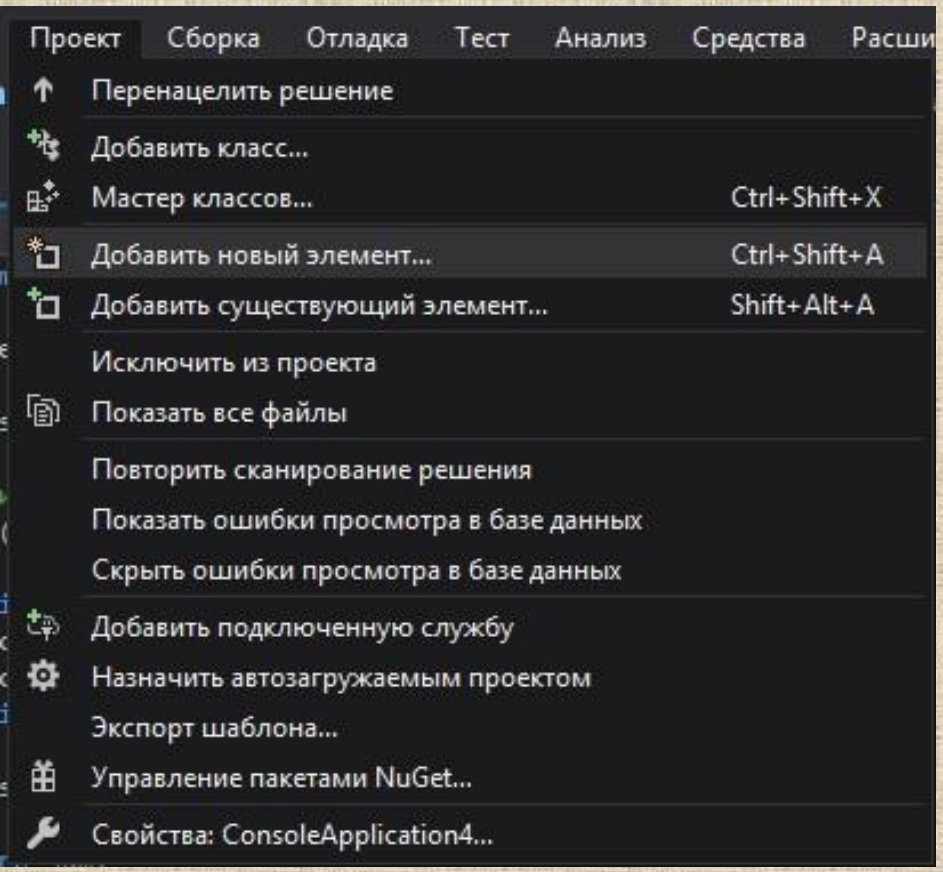
2 способ:

Если же функция слишком объемная, или вообще этих функций несколько, то можно прямо перед main() прописать только прототипы функций, а сами функции описывать уже после main();

```
#include <iostream>
using namespace std;

int myFunction(); // Прототип функции

int main()
{
    int itog = myFunction();

    system("pause");
}

int myFunction()
{
    int x = 5;
    int y = x + 25;
    return y;
}
```

Прототип функции – это краткое описание функции, которое располагается перед main() и содержит в себе только имя функции, тип возвращаемого значения, количество аргументов и их типы данных. Само описание функции переносится в раздел после main().

3 способ:



Еще одним вариантом расположения функции является использование заголовочных файлов. Чтобы создать заголовочный файл, нужно в главном меню IDE Visual Studio выбрать пункт «Проект» и щелкнуть левой кнопкой мыши на пункте «Добавить новый элемент». Или просто нажать на клавиатуре сочетание клавиш

Далее нужно выбрать «Файл заголовка(.h)»; изменить имя файла таким образом, чтобы по имени было возможно определить, каков функционал данного заголовочного файла; указать папку, в которой будет создан этот заголовочный файл.

После этого необходимо нажать на «Добавить», и в папке с проектом появится созданный заголовочный файл.

В файле, где находится main() нужно этот заголовочный файл подключить, используя ключевое слово #include. Например:

#include “имяФайла.h”

Функции, вызывающие друг друга

Рассмотрим простой пример, в котором реализована следующая последовательность выполнения блоков программы: сначала из функции main() вызовем функцию Sum(), которая в свою очередь вызовет еще одну функцию Square(). Square() будет возвращать квадрат числа в Sum(), а функция Sum() будет суммировать квадраты чисел. После выполнения своей задачи функция Sum() вернет в main() результат. Нужно понимать, что вызванная функция может вернуть свое значение только в ту функцию, из которой она была вызвана. В нашем примере не может быть возврата по прямому пути из функции Square() в функцию main().

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

int main()
{
    setlocale(LC_ALL, "rus");

    double firstNum = 0;
    double secondNum = 0;
    cout << "Программа вычисляет сумму квадратов чисел.\nВведите 1-е число: ";
    cin >> firstNum;
    cout << "Введите второе число: ";
    cin >> secondNum;

    cout << "Ответ: " << Sum(firstNum, secondNum) << endl;
}
```

```cpp
#include "Square.h"
using namespace std;

double Sum(double firstNumber, double secondNumber)
{
    return (Square(firstNumber)) + (Square(secondNumber));
}
```

```cpp
using namespace std;

double Square(double number)
{
    return number * number;
}
```

В нашем примере все три функции реализованы в разных файлах.

Результат работы

```
Программа вычисляет сумму квадратов чисел.
Введите 1-е число: 2
Введите второе число: 3
Ответ: 13
```

Если в рассмотренной нами программе описывать все функции в одном файле, то необходимо соблюдать следующий порядок размещения функций. Сначала мы должны описать функцию до ее вызова. Поэтому первой функцией в описании была бы Square(), второй – Sum(), последней – main().

Если же нами принято решение использовать прототипы функций, то для нашей программы порядок расположения прототипов и самих функций был бы произвольным.

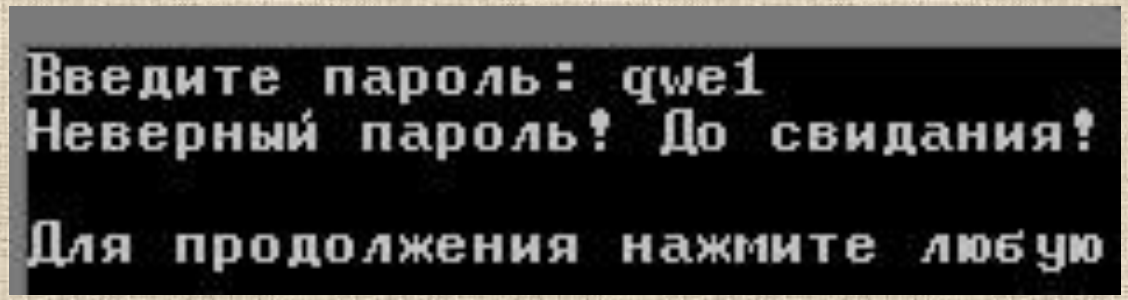
Формальные и фактические параметры

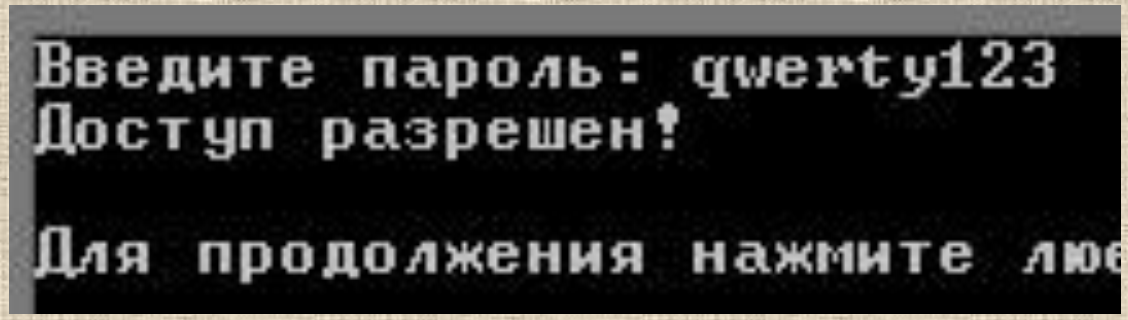
Как видно из примера на предыдущих слайдах, в каждую из вызываемых функций мы передавали некоторые аргументы (параметры). Параметры, которые передаются в функцию при ее вызове, называются фактическими параметрами. А параметры, которые принимает вызываемая функция, называются формальными параметрами.

В вызываемой функции создаются дополнительные переменные (в скобках), в которые копируются значения из вызывающей функции. Поэтому для формальных параметров необходимо указывать типы данных.

Функция в результате своей работы может вернуть не более одного значения

Задача 1. Написать программу с функцией проверки ввода пароля. Использовать тип возвращаемого значения и тип аргумента функции – string.





Задача 2. Написать программу, в которой функция принимает в качестве параметров 2 числа, введенные пользователем. Эта функция сравнивает полученные числа и возвращает в main() знак >, < или =.
Ввод/вывод данных производится в функции main().