Подпрограммы в программировании

Что это такое?

Подпрограммой (функцией) обычно называют какую-либо обособленную последовательность действий

Например:

- Функция возведения числа в степень;
- Функция вывода текущего времени;
- Функция переворота строки;
- Функция сортировки;

Функции могут принимать на вход какие-либо значения (функция возведения числа в степень - число и степень), либо работать без них (функция вывода текущего времени).

Функции могут возвращать какое-либо значение, либо не возвращать.

> *Когда-то подпрограммы, которые **не возвращали** значения, называли процедурами, а подпрогрограммы, которые что-то **возвращали** - функциями.*

> *Теперь, всё называют функциями **:)***

Функции в C++

Рассмотрим на примере простой функции `printHello()`:

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

void printHello() {
    cout << "Hello, World!" << endl;
}

using namespace std;
int main() {
    printHello();
    return 0;
}
```

Мы объявили функцию вне главной функции `main`. Результатом работы функции будет вывод на консоль `Hello, world!`.

Давайте познакомимся с ней поближе:

- **Ключевое слово** `void` говорит там о том, что данная функция ничего не возвращает;
- `printHello` - **имя** нашей функции. Правила именования такие же, как и для переменных.

> *Кстати, хорошим стилем считается в названии функции использовать глаголы, описывающие происходящее в этой функции!*

- После имени функции **скобки** `()` пустые, а значит, наша функция не принимает на вход какие-либо значения.

Давайте сделаем нашу функцию дружелюбнее:

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

void printHello(string name) {
    cout << "Hello, " << name << "!" << endl;
}

using namespace std;
int main() {
    string myName = "Andrew";
    printHello(myName);
    return 0;
}
```

Теперь у нашей функции появился входной параметр `name` типа `string`. Результат работы: `Hello, Andrew!`

Мы можем запустить функцию с любой строкой!

Использовать вывод на консоль в функции считается небезопасным. Как насчет того, чтобы просто вернуть сообщение из функции? Давайте снова улучшим её:

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

string getHelloMessage(string name) {
    string resultMessage = "Hello, " + name  + "!";
    return resultMessage;
}

using namespace std;
int main() {
    string myName = "Andrew";
    string message = getHelloMessage(myName);
    cout << message << endl;
    return 0;
}
```

Теперь вместо `void` у функции стоит `string`. Кстати, теперь функции лучше подойдет имя `getHelloMessage`. В функции `main` мы вызываем функцию и записываем её выходное значение в переменную `message`.

Так, у нас получилась забавная, однако, бесполезная функция.

Как насчет того, чтобы написать функцию, которая ищет наибольший общий делитель (далее - НОД) двух целых чисел?

Поехали!

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

// Функция поиска наибольшего делителя положительных чисел a и b
int getMaxCommonDivider(int a, int b) {
    int commonDivider = 1;
    int minNumber;

    if (a > b)
        minNumber = b;
    else
        minNumber = a;

    for (int i = 1; i <= minNumber; i++)
        if ((a % i == 0) && (b % i ==0))
            commonDivider = i;

    return commonDivider;
}

using namespace std;
int main() {
    int a = 18;
    int b = 27;
    cout << getMaxCommonDivider(a, b) << endl; // 9
    return 0;
}
```

Мы создали функцию `getMaxCommonDivider`, у которой есть два входных параметра типа `int`. Данная функция возвращает значение типа `int` - НОД входных параметров.

Сравните:

```
...
int main() {
    int a = 18;
    int b = 27;
    cout << getMaxCommonDivider(a, b) << endl; // 9
    return 0;
}
```

```
...
int main() {
    int a = 18;
```

```
    int b = 27;
    int commonDivider = 1;
    int minNumber;

    if (a > b)
        minNumber = b;
    else
        minNumber = a;

    for (int i = 1; i <= minNumber; i++)
        if ((a % i == 0) && (b % i ==0))
            commonDivider = i;

    cout << commonDivider << endl; // 9
    return 0;
}
```

Согласитесь, первый вариант при использовании смотрится лучше.

Преимущества:

- улучшается читаемость кода;

- вы можете использовать функции других разработчиков без необходимости понимания того, что в них происходит;

- если в функция работает неправильно, требуется лишь одно исправление - в теле функции, а не во всех файлах, где высчитывается НОД.

Вывод

Зачастую в коде требуется *выполнение одних и тех же действий*.

В целях *улучшения читаемости* и *сокращения времени отладки* программ эти действия можно сохранять в функции.

Выбрав *входные параметры* и *возвращаемое значение*, вы *снизите вероятность ошибки*, *улучшите структуру приложения* и, тем самым, *упростите процесс разработки себе и вашим коллегам*!