Функции. Указатели. Возможности функций в C++

Функции

Функция в программировании — это вспомогательный алгоритм (подпрограмма), предназначенный для решения конкретной задачи.

Предназначение функций:

- Разделение кода на части.
- Избавление от дублирования кода.
- Выделение общих блоков.

Синтаксис:

```
тип_возвр_знач имя_функц([параметры])
{
     //тело функции
     return [значение];
}
```

где тип_возвр_знач — это тип данных возвращаемого функцией значения;

имя_функц — это идентификатор;

[параметры] — необязательны, записываются через запятую в формате тип имя;

тип_возвр_знач имя_функц([параметры]) — это заголовок (прототип) функции

//тело функции — все, что находится в функции между {}

return [значение]; — команда, завершающая выполнение функции и возвращающая значение. Тип значения должен совпадать стип_возвр_знач. Код, написанный после вызова этой команды, выполняться не будет!

Если функция не должна возвращать значение, вместо тип_возвр_знач пишем **void**. Тогда команду return можно не писать. Если же необходимо завершить выполнение такой функции досрочно, то пишется return; без значения.

Функции пишутся вне любой другой функции. Поскольку main это тоже функция, другие функции пишутся либо до нее, либо после нее. Если в функции вызывается другая функция, которая описана после нее по коду, то до описания тела этой функции должен быть скопирован ее заголовок, в конце которого нужно поставить ;

Пример: см. задачу 1 в https://replit.com/@IekatierinaLat1/Functions#main.cpp

Указатели

Указатель — это адресная переменная. В ней хранится адрес участка памяти, в которой может храниться значение переменной. В C++ существует три типа указателей: на объект, на функцию и на void. Обычно если говорят просто «указатель», подразумевают указатель на объект. Их мы и рассмотрим. Синтаксис:

```
тип *имя
```

Операции с указателями:

- ++ — инкрементирование (увеличение) адреса на единицу соответствующего типа. Если указатель имеет тип int с размером 4 байта, значение переменной также увеличится на 4.

- — (два минуса) — декрементирование (уменьшение). Работает аналогично предыдущему пункту.
- + константа и – константа — увеличение или уменьшение на константу. Работает аналогично первым двум пунктам.
- & — операция взятия адреса (адресации). Позволяет работать с обычной переменной как с указателем.
- * — операция разадресации. Доступ к значению через указатель. Позволяет работать со значением указателя.
- new — выделение памяти под указатель.
- delete — освобождение памяти, занятой указателем.

Применение указателей:

- Передача параметров в функцию.
- Работа с массивами.
- Динамические переменные и структуры данных (в этом семестре не рассматривается).

Примеры: https://replit.com/@IekatierinaLat1/Pointers#main.cpp

Передача параметров в функцию

В C++ есть следующие способы передачи параметров в функцию:

- по значению тип имя (передается копия, значение не меняется);
- по адресу тип *имя (передается адрес, значение меняется);
- по ссылке тип &имя (передается ссылка, значение меняется).

Ссылка — синоним имени (дополнительное имя для существующей переменной).

Пример: см. задачу 2 в https://replit.com/@IekatierinaLat1/Functions#main.cpp

Знакомство с рекурсией

Рекурсивной называется функция, вызывающая сама себя. Чтобы этот процесс не продолжался бесконечно, в функции обязательно прописывается условие ее завершения. Во многих задачах условие завершения пишется в самом начале функции.

Пример: Факториал числа n определяется как произведение чисел от 1 до n $n! = 1 \cdot 2 \cdot .. \cdot (n-1) \cdot n$

При этом 1!=1, а 0!=1 (доопределяется для удобства).

Также факториал можно задать с помощью следующей формулы $a_n = a_{n-1} * n$, где $a_1 = 1$

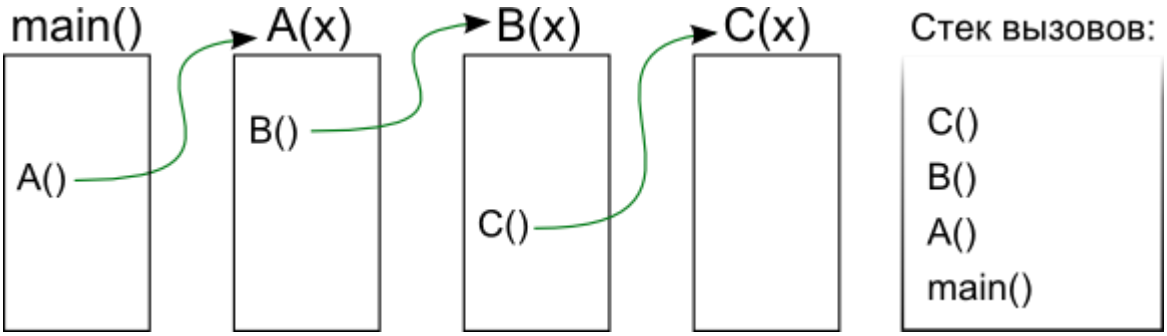
Пример: см. задачу 3 в https://replit.com/@IekatierinaLat1/Functions#main.cpp

Стек вызовов

Рекурсия возможна благодаря тому, что в программе используется стек вызовов.

Стек — это структура данных, реализующая принцип «последним зашел, первым ушел» (LIFO, Last In First Out). Стек можно сравнить с железнодорожным тупиком. Чтобы "вытащить" из него вагон, необходимо сначала убрать все вагоны, которые стоят перед ним.

При вызове функции ее адрес, параметры и некоторая другая информация помещаются в стек вызовов. При завершении функции информация о ней извлекается из стека. Использование стека вызовов гарантирует возвращение в нужную точку программы.



Аргументы (параметры) по умолчанию

В некоторых случаях удобно задавать параметрам функции значения по умолчанию. Если параметр не указывается при передаче в функцию, то его значение заменяется значением по умолчанию. Если же этот параметр передается в функцию, то значение заменяется на переданное.

В C++ параметры со значениями по умолчанию обязательно должны находиться в конце списка параметров функции.

Пример: см. задачу 1 в https://replit.com/@IekatierinaLat1/functions#main.cpp

Перегрузка функций

Перегрузка функций — возможность использовать одно и то же имя для функций, выполняющих схожие, но не одинаковые действия в зависимости от переданных параметров. Широко используется в ООП (см. далее).

Пример: см. задачу 2 в https://replit.com/@IekatierinaLat1/functions#main.cpp

Шаблоны функций

Шаблоны функций в C++ позволяют создавать единые функции для данных разных типов. Шаблоны удобны при организации различных алгоритмов, например алгоритмов работы с массивами.

Пример: см. задачу 3 в https://replit.com/@IekatierinaLat1/functions#main.cpp

Статические переменные

Статическая переменная сохраняет значение, полученное при предыдущем вызове функции. При первоначальном вызове функции статическая переменная принимает значение инициализатора, поэтому должна обязательно инициализироваться в теле функции.

Пример: см. задачу 4 в https://replit.com/@IekatierinaLat1/functions#main.cpp

Правильное использование рекурсии

Рекурсивная запись алгоритма часто выглядит более короткой, однако далеко не всегда использование рекурсии является наилучшим выходом.

Числа Фибоначчи — элементы числовой последовательности, в которой первые два числа равны 0 и 1, а каждое последующее число равно сумме двух предыдущих чисел

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, ...

Иначе эту последовательность можно записать с помощью формулы

$a_n=a_{n-1}+a_{n-2}$, где $a_0=0$ и $a_1=1$.

Поскольку последовательность задается рекуррентно, возникает естественное желание использовать рекурсию для решения задачи. Первый пришедший в голову вариант обычно выглядит так, как показано в примере, функция называется badFib.

Почему такая реализация является наихудшим решением?

Для нахождения очередного числа мы вычисляем два предыдущих отдельно друг от друга. Для каждого из них мы заново вычисляем два предыдущих и т.д. Таким образом, для вычисления n-го числа Фибоначчи нам приходится выполнить почти 2^n операций. Иначе говоря, сложность этого алгоритма равна $O(2^n)$, то есть алгоритм имеет экспоненциальную сложность.

Вторая реализация намного эффективнее, так как она задействует лишь один рекурсивный вызов, который по сути является альтернативным вариантом записи простого циклического алгоритма. Вычислительная сложность равна O(n).

Пример: см. задачу 5 в https://replit.com/@IekatierinaLat1/functions#main.cpp

Передача имен функций в качестве параметров (материал повышенной сложности)

В C++ можно передавать в функцию другую функцию в качестве параметра. В таком случае используются указатели на функцию.

Пример: см. задачу 6 в https://replit.com/@IekatierinaLat1/functions#main.cpp