Модуль 7

- Параметры и аргументы функций
- Передача аргументов по ссылке/значению
- Перегрузка функций
- Параметры по умолчанию
- Стек и куча

Параметры и аргументы функций

Параметр функции — это переменная, которая используется в функции, и значение которой предоставляет вызывающий объект. Параметры указываются при объявлении функции в круглых скобках. Если их много, то они перечисляются через запятую.

```
int add(int a, int b)
{
    return a + b;
}
```

Аргумент функции — это значение, которое передается из caller-а в функцию и которое указывается в скобках при вызове функции в вызывающий объект:

```
add(4, 5); // 4 и 5 – это аргументы функции add()
```

Параметры и аргументы функций

Способы передачи аргументов в функцию:

1. передача по значению;
2. передача по ссылке;
3. передача по адресу.

Параметры и аргументы функций

Плюсы передачи по значению:

- Аргументы, переданные по значению, могут быть переменными (например, `x`), литералами (например, `8`), выражениями (например, `x + 2`), **структурами**, классами или **перечислителями**.
- Аргументы никогда не изменяются функцией, в которую передаются, что предотвращает возникновение **побочных эффектов**.

Минусы передачи по значению:

- Копирование структур и классов может привести к значительному снижению производительности (особенно, когда функция вызывается много раз).

Когда использовать передачу по значению:

- При передаче фундаментальных типов данных и перечислителей, когда предполагается, что функция не должна изменять аргумент.

Когда не использовать передачу по значению:

- При передаче **массивов**, структур и классов.

Параметры и аргументы функций

Плюсы передачи по ссылке:

- Ссылки позволяют функции изменять значение аргумента, что иногда полезно.
- Более эффективная передача из-за отсутствия копирования параметра.
- Ссылки могут использоваться для возврата сразу нескольких значений из функции (через параметры вывода).

Минусы передачи по ссылке:

- Трудно определить, является ли параметр, переданный по неконстантной ссылке, параметром ввода, вывода или того и другого одновременно.
- По вызову функции невозможно определить, будет аргумент изменен функцией или нет.

Когда использовать передачу по ссылке:

- при передаче структур или классов (используйте const, если нужно только для чтения);
- когда нужно, чтобы функция изменяла значение аргумента.

Когда не использовать передачу по ссылке:

- при передаче фундаментальных типов данных (используйте передачу по значению);
- при передаче обычных массивов (используйте передачу по адресу).

Параметры и аргументы функций

Плюсы передачи по адресу:

- Передача по адресу позволяет функции изменить значение аргумента, что иногда полезно.
- Высокая скорость передачи больших **структур** или классов.
- Возможно вернуть сразу несколько значений из функции.

Минусы передачи по адресу:

- Все указатели нужно проверять, не являются ли они нулевыми.
- Более медленный доступ к переменным из-за необходимости разыменовывать указатель.

Когда использовать передачу по адресу:

- при передаче обычных массивов,
- если нужна арифметика указателей,
- При необходимости передавать null.

Когда не использовать передачу по адресу:

- при передаче структур или классов (используйте передачу по ссылке);
- при передаче фундаментальных типов данных (используйте передачу по значению).

Параметры по умолчанию

Параметр по умолчанию (или ***«необязательный параметр»***) — это параметр функции, который имеет определенное (по умолчанию) значение.

```
#include <iostream>

void printValues(int a, int b=5)
{
    std::cout << "a: " << a << '\n';
    std::cout << "b: " << b << '\n';
}
int main()
{
    printValues(1);
    printValues(6, 7);
}
```

Параметры по умолчанию

```
void openLogFile(std::string filename="default.log");


void printStringInColor(std::string str, Color color=COLOR_RED);


void printValues(int a=10, int b=11, int c=12)
{
    std::cout << "Values: " << a << " " << b << " " << c << '\n';
}
```

Параметры по умолчанию

- Все параметры по умолчанию в прототипе или в определении функции должны находиться справа.
- Если имеется более одного параметра по умолчанию, то самым левым параметром по умолчанию должен быть тот, который с наибольшей вероятностью (среди всех остальных параметров) будет явно переопределен пользователем.
- Объявляйте параметры по умолчанию в предварительном объявлении функции, в противном случае (если функция не имеет предварительного объявления) — объявляйте в определении функции.

Перегрузка функций

Перегрузка функций — это возможность определять несколько функций с одним и тем же именем, но с разными параметрами.

```
int subtract(int a, int b)
{
    return a - b;
}
double subtract(double a, double b)
{
    return a - b;
}
int subtract(int a, int b, int c)
{
    return a - b - c;
}
```

Перегрузка функций

При компиляции перегруженной функции, C++ выполняет следующие шаги для определения того, какую версию функции следует вызывать:

Шаг №1: **C++ пытается найти точное совпадение**.

```
1 void print(char *value);
2 void print(int value);
3
4 print(0); // точное совпадение с print(int)
```

Перегрузка функций

Шаг №2: **Если точного совпадения не найдено, то C++ пытается найти совпадение путем дальнейшего неявного преобразования типов**.

- ➔ char, unsigned char и short конвертируются в int;
- ➔ unsigned short может конвертироваться в int или unsigned int (в зависимости от размера int);
- ➔ float конвертируется в double;
- ➔ enum конвертируется в int.

```
1 void print(char *value);
2 void print(int value);
3
4 print('b'); // совпадение с print(int) после неявного преобразования
```

Перегрузка функций

Шаг №3: **Если неявное преобразование невозможно, то C++ пытается найти соответствие посредством стандартного преобразования**.

- ➔ Любой числовой тип будет соответствовать любому другому числовому типу, включая unsigned (например, int равно float).
- ➔ enum соответствует формальному типу числового типа данных (например, enum равно float).
- ➔ Ноль соответствует типу указателя и числовому типу (например, `0` как `char *` или `0` как `float`).
- ➔ Указатель соответствует указателю типа void.

```
1 struct Employee; // определение упустим
2 void print(float value);
3 void print(Employee value);
4
5 print('b'); // 'b' конвертируется в соответствие версии print(float)
```

Перегрузка функций

Шаг №4: **C++ пытается найти соответствие путем пользовательского преобразования**.

```
1 class W; // с пользовательским преобразованием в тип int
2
3 void print(float value);
4 void print(int value);
5
6 W value; // объявляем переменную value типа класса W
7 print(value); // value конвертируется в int и, следовательно, соответствует print(int)
```

Перегрузка функций

Что делать если есть несколько совпадений?

- Определить новую перегруженную функцию, которая принимает параметры именно того типа данных, который вы используете в вызове функции.
- Явно преобразовать с помощью операторов явного преобразования неоднозначный параметр(ы) в соответствии с типом функции, которую вы хотите вызвать. Например, чтобы вызов `print(0)` соответствовал `print(unsigned int)`, вам нужно сделать следующее.

```
print(static_cast<unsigned int>(0));
```

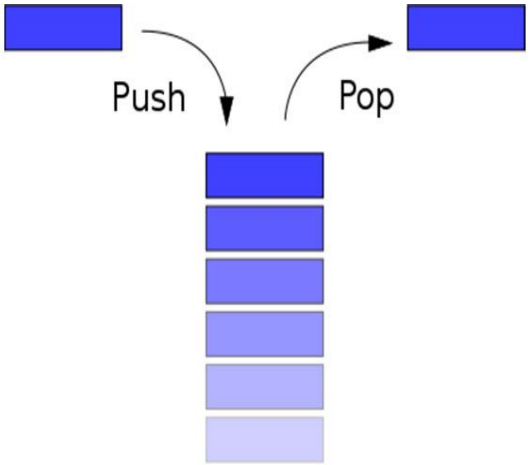
Стек и куча

Типы памяти

- **Статическая память:** Это область памяти, которая выделяется при запуске программы и освобождается только при завершении работы программы. Сюда относятся глобальные переменные и статические переменные в функциях.
- **Стековая память:** Это область памяти, используемая для хранения локальных переменных и вызовов функций. Переменные в стеке создаются при входе в блок кода и уничтожаются при выходе из него. Это обеспечивает быстрый доступ к переменным, но размер стека ограничен.
- **Динамическая память/Куча**: Это область памяти, которую можно выделить во время выполнения программы с помощью оператора `new` и освободить с помощью оператора `delete`. Динамическая память используется для создания объектов переменной длины или для временного выделения больших блоков памяти.

Стек и куча

Стек и куча



Стек — это структура данных типа **LIFO** (англ. *«**L**ast **I**n, **F**irst **O**ut» = «Последним пришел, первым ушел»*).

Переполнение стека (англ. ***«stack overflow»***) происходит, когда запрашиваемой памяти нет в наличии (вся память уже занята).

Стек и куча

Преимущества и недостатки стека:

- Выделение памяти в стеке происходит сравнительно быстро.
- Память, выделенная в стеке, остается в области видимости до тех пор, пока находится в стеке. Она уничтожается при выходе из стека.
- Вся память, выделенная в стеке, обрабатывается во время компиляции, следовательно, доступ к этой памяти осуществляется напрямую через переменные.
- Поскольку размер стека является относительно небольшим, то не рекомендуется делать что-либо, что съест много памяти стека (например, передача по значению или создание локальных переменных больших массивов или других *затратных* структур данных).

Стек и куча

Куча отслеживает память, используемую для динамического выделения.

Стек и куча

Преимущества и недостатки кучи:

- Выделение памяти в куче сравнительно медленное.
- Выделенная память остается выделенной до тех пор, пока не будет освобождена (остерегайтесь утечек памяти) или пока программа не завершит свое выполнение.
- Доступ к динамически выделенной памяти осуществляется только через указатель. Разыменование указателя происходит медленнее, чем доступ к переменной напрямую.
- Поскольку куча представляет собой большой резервуар памяти, то именно она используется для выделения больших массивов, структур или классов.