Алгоритмизация и программирование

Лекция 3

Типы

```cpp
int count;
std::string str;
```

```cpp
struct Human{
    int age;
    std::string name;
    double salary;
};
Human john;
```

```cpp
int nums[100];
std::array<int, 100> nums2();
std::vector<std::string> text(100);
```

```cpp
class PartyHuman{
    // состояние
    int age;
    std::string name;
    double salary;
    // Поведение (интерфейс)
public:
    void hang_out();
};
PartyHuman slurms;
```


```cpp
std::tuple<std::string, int, int, bool> params;
```

```cpp
std::pair<std::string, int> login;
```

Учебная задача 1

Пользователь вводит ссылку в формате Markdown. Ссылка может быть обычная или на изображение:

- `[текст ссылки](адрес ссылки)`
- `![пояснение](адрес изображения)`

Преобразуйте и распечатайте на экран ссылку в формате HTML:

- `<a href="адрес ссылки">текст ссылки</a>`
- `<img alt="пояснение" src="адрес изображения">`

Гарантируется, что круглые и квадратные скобки текста или адреса. На вход может быть подана строка не соответствующая ссылке в формате Markdown, в этом случае напечатайте: NO.

https://wandbox.org/permlink/PyLCkbX5N17jpRrx

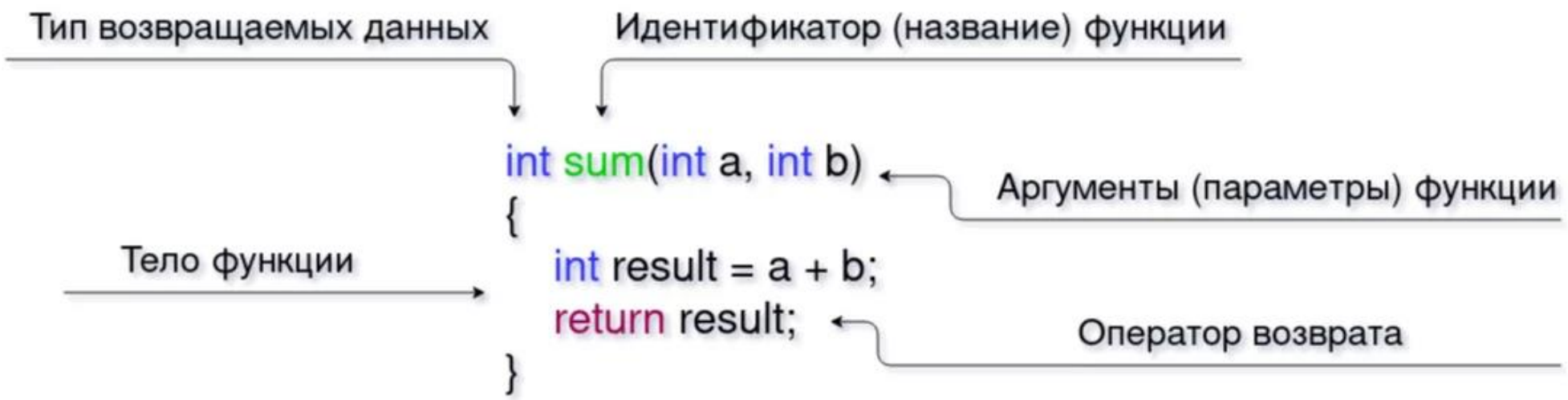
Проблемы решения Задачи 1

- Большой размер функции, приходится скролить;
- Без комментариев сходу непонятно, что делает код;
- Нужно постоянно вносить изменения в код функции, при изменении условий. Или создавать множество почти одинаковых копий;
- Нет возможности протестировать отдельную часть функционала, только всё вместе.

Принцип единой ответственности

Принцип единой ответственности (Single Responsibility Principle) – это **принцип**, который гласит, что каждый модуль, класс или **функция** в компьютерной программе должны нести **ответственность** за одну часть функциональности этой программы, и она должна инкапсулировать эту часть.

Функция


Тип возвращаемых данных
Идентификатор (название) функции
int sum(int a, int b)
{
int result = a + b;
return result;
}
Аргументы (параметры) функции
Тело функции
Оператор возврата

Функция

Объявление (declaration) функции вводит имя функции и ее тип в область видимости (scope);

```
int sum(int a, int b);
```

```
auto sum(int a, int b) -> int;
```

Определение (definition) функции связывает имя/тип функции с её телом;

```
int sum(int a, int b)
{
    int result = a + b;
    return result;
}
```

```
auto sum(int a, int b) -> int
{
    int result = a + b;
    return result;
}
```

https://wandbox.org/permlink/DAISkDauoPcyNOdF

Объявление функции

Прототипом функции в языке Си или C++ называется объявление функции, не содержащее тела функции, но указывающее имя функции, арность, типы аргументов и тип возвращаемых данных.

```
int sum(int a, int b);
```

Сигнатура функции – это части прототипа функции, которые компилятор использует для выполнения разрешения перегрузки.

```
sum(int, int);
```

Формальные параметры (параметры) – это собственно параметры указанные в прототипе/сигнатуре функции (в данном случае `a` и `b`).

Вызов функции

Вызов функции - передача управления потоком исполнения команд в другую точку программы с последующим возвратом в точку вызова.

```
int main()
{
    auto res = sum(2, 2);
    std::cout << res << std::endl;
}
```

Фактические параметры (аргументы) – конкретные значения, которые передаются формальным параметрам (в данном случае 2 и 2).

Параметры

Передача данных **по значению.** Создаёт локальную копию передаваемых данных.

```
void swap(int a, int b){
    int t = a;
    a = b;
    b = t;
}
```

Передача данных **по ссылке.** Создаёт дополнительное имя для переменной переданной в качестве аргумента.

```
void swap(int& a, int& b){
    int t = a;
    a = b;
    b = t;
}
```

Передача данных **по указателю.** Создаёт копию, но не данных, а адреса по которому они находятся.

```
void swap(int* a, int* b){
    int t = *a;
    *a = *b;
    *b = *t;
}
```

const

Квалификатор **const** запрещает изменять параметры.

```
void swap(const int a, const int b){
    int t = a;
    a = b;
    b = t;
}
```

Параметры функции main

Без параметров

```
int main();
```

Доступ к параметрам запуска программы

```
int main(int argc, char const *argv[]);
```

Доступ к параметрам запуска и переменным окружения

```
int main(int argc, char* argv[], char* envp[]);
```

Оператор return

Оператор **return** осуществляет прерывание исполнения текущей функции и возврат потока исполнения в точку вызова.

Для void функций не обязателен. Функция завершится после выполнения последней команды в теле функции.

Для не void функций обязателен. После оператор return должно быть указано значение того же (или приводимое) типа, что и в прототипе. Это значение вернётся в качестве результата в вызывающую функцию.

В функции main разрешено не указывать. В этом случае результат будет 0.

Может присутствовать в теле функции множество раз.

```
int sum(int a, int b)
{
    int result = a + b;
    return result;
}
```

```
void print_hello(){
    std::cout << "Hello" << std::endl;
}
```

Возвращаемое значение

Возврат данных **по значению.** Создаёт копию возвращаемых данных и отдаёт наружу.

```
int sum(int a, int b){
    int result = a + b;
    return result;
}
```

Возврат данных **по ссылке.** Даёт доступ в нижнему коду к локальной переменной функции.

```
int& sum(int a, int b){
    int result = a + b;
    return result;
}
```

Возврат данных **по указателю.** Передаёт наружу информацию об адресе, по которому лежат данные.

```
int* sum(int a, int b){
    int result = a + b;
    return &result;
}
```

Время жизни и область видимости локальных переменных

Область видимости локальных переменных, в том числе и параметров. От точки объявления до конца области видимости. Конец области видимости определяется либо концом функции либо концом блока.

Локальные переменный функции, в том числе и параметры, живут от момента создания до момента выхода из области видимости. Кроме static переменных.

const

Квалификатор **const** не играет роли если возврат по значению. В остальных случаях запрещает изменение данных.

```
const int sum(int a, int b){
    int result = a + b;
    return result;
}
```

Значение возвращаемое main

Без параметров

```
int main();
```

Доступ к параметрам запуска программы

```
int main(int argc, char const *argv[]);
```

Доступ к параметрам запуска и переменным окружения

```
int main(int argc, char* argv[], char* envp[]);
```