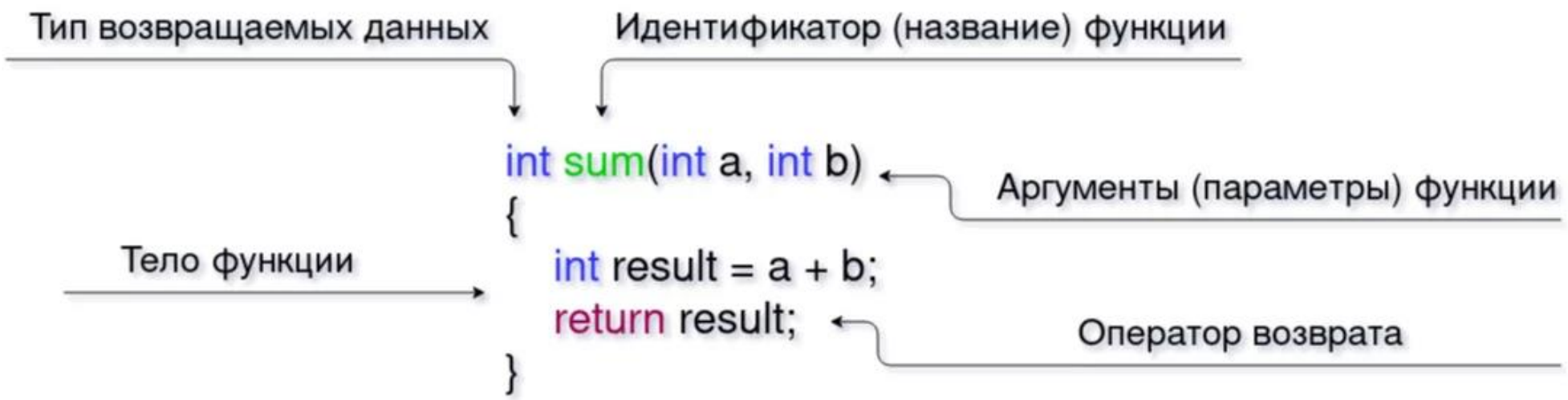
Алгоритмизация и программирование

Лекция 5

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).

Код внутри функции

Время жизни и область видимости локальных переменных

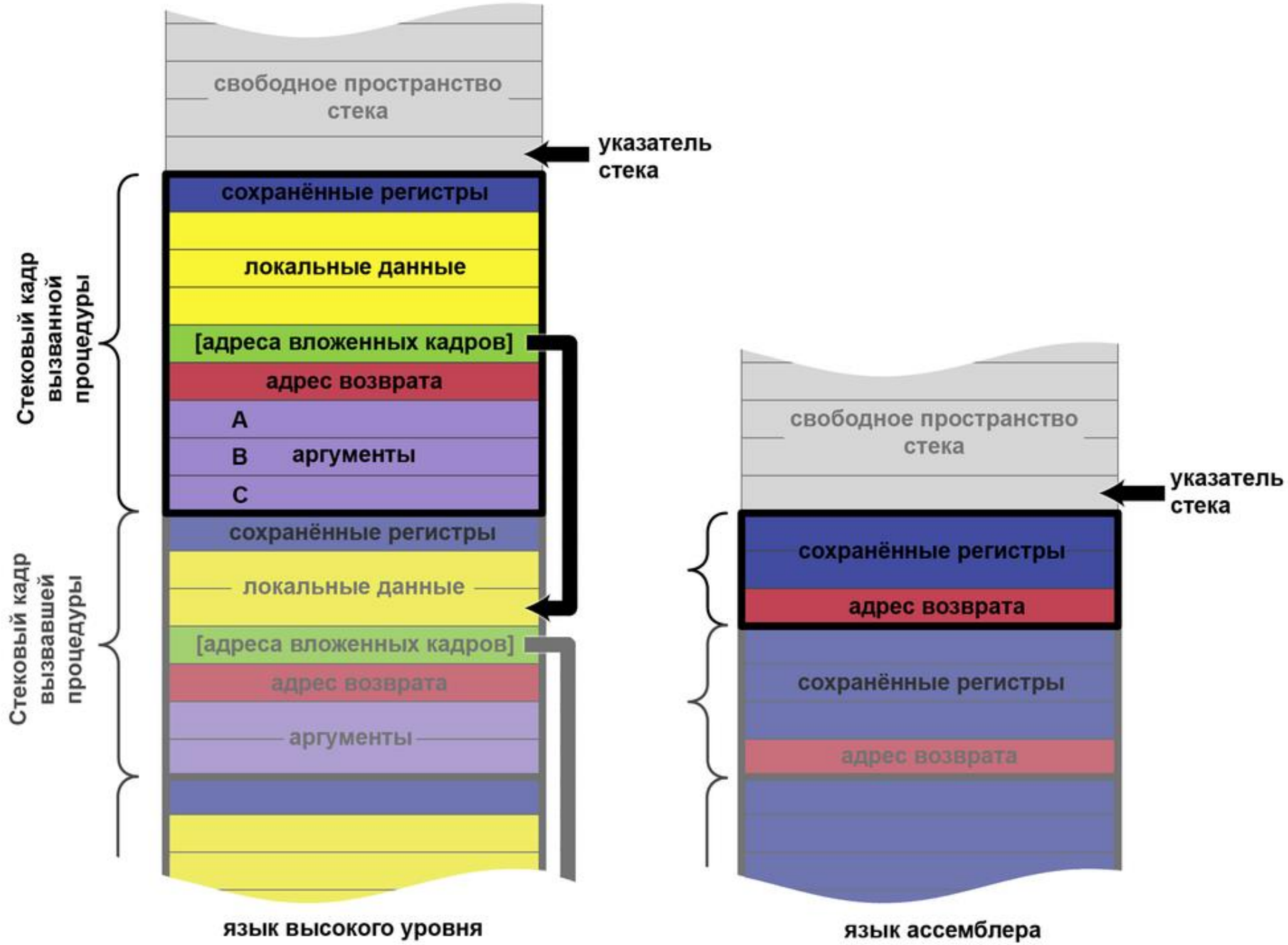
Область видимости локальных переменных, в том числе и параметров. От точки объявления до конца области видимости. Конец области видимости определяется либо концом функции либо концом блока.

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

Стек вызова функций

Стек вызова функций

Стек вызовов (стек) может использоваться для различных нужд, но основное его назначение — отслеживать место, куда каждая из вызванных процедур должна вернуть управление после своего завершения. Для этого при вызове процедуры (командами **вызова**) в **стек** заносится адрес команды, следующей за командой **вызова** («адрес возврата»).


свободное пространство стека
указатель стека
сохранённые регистры
локальные данные
[адреса вложенных кадров]
адрес возврата
A
B
C
аргументы
Стековый кадр вызванной процедуры
сохранённые регистры
локальные данные
[адреса вложенных кадров]
адрес возврата
аргументы
Стековый кадр вызвавшей процедуры
язык высокого уровня
свободное пространство стека
указатель стека
сохранённые регистры
адрес возврата
сохранённые регистры
адрес возврата
язык ассемблера

Передача данных в функцию

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 const *argv[], char const *envp[]);
```

Получение данных из функции

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;
}
```

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

Согласно стандарту тип возвращаемого значения функции `main` должен быть только `int`.

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

Только для функции `main` разрешается не указывать оператор `return`, для всех остальных не `void` функций `return` обязателен. В случае, если в `main` нет оператора `return`, то гарантируется, что она вернёт 0.
Значение, которое возвращает функция `main` передаётся операционной системе как результат работы программы.
Это значение в общем случае не влияет ни на что, но его можно использовать в shell-скриптах.
По соглашению если программа вернёт 0, то считается, что она завершилась корректно, а любые другие значения – это не корректное завышение программы. При этом значение для каждого кода ошибки разработчик придумывает по своему желанию (и описывает в документации).