Вводное занятие. Основы устройства памяти и выполнения программы. Шаблонные функции

Мстоян Амиран

Сегментация памяти

- Оперативная память, используемая в программе на C++, разделена на области двух типов:
  1. сегменты данных,
  2. сегменты кода (текстовые сегменты).
- В сегментах кода содержится код программы.
- В сегментах данных располагаются данные программы (значения переменных, массивы и пр.).
- При запуске программы выделяются два сегмента данных:
  1. сегмент глобальных данных,
  2. стек (для локальных переменных).
- В процессе работы программы могут выделяться и освобождаться дополнительные сегменты памяти
- Обращения к адресу вне выделенных сегментов — ошибка времени выполнения (access violation, segmentation fault).

Как выполняется программа?

- Каждой функции в скомпилированном коде соответствует отдельная секция.
- Адрес начала такой секции — это адрес функции.
- Телу функции соответствует последовательность команд процессора.
- Работа с данными происходит на уровне байт, информация о типах отсутствует.
- В процессе выполнения адрес следующей инструкции хранится в специальном регистре процессора IP (Instruction Pointer).
- Команды выполняются последовательно, пока не встретится специальная команда (например, условный переход или вызов функции), которая изменит IP.

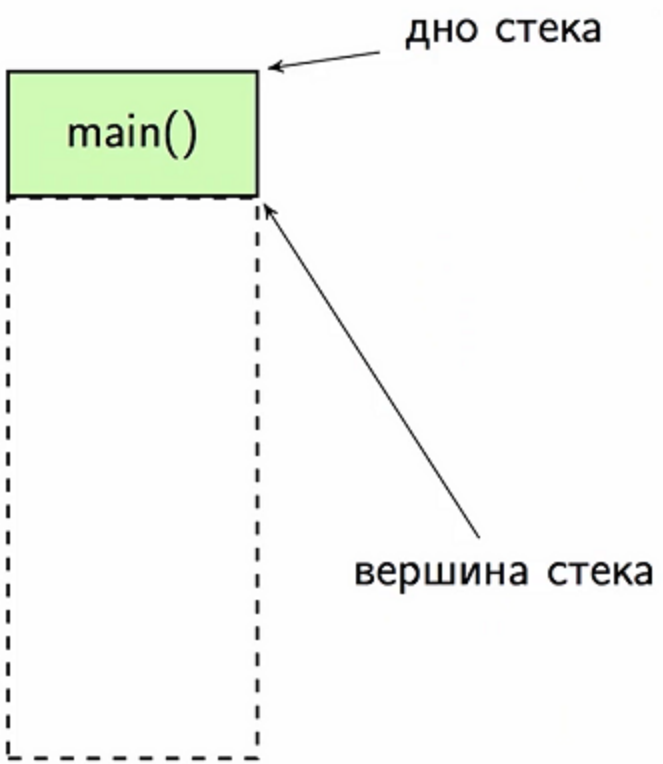
Ещё раз о линковке

- На этапе компиляции объектных файлов в места вызова функций подставляются имена функций.
- На этапе линковки в места вызова вместо имён функций подставляются их адреса.
- Ошибки линковки:
  1. `undefined reference`
     Функция имеет объявление, но не имеет тела.
  2. `multiple definition`
     Функция имеет два или более определений.
- Наиболее распространённый способ получить `multiple definition` — определить функцию в заголовочном файле, который включён в несколько `.cpp` файлов.

Стек вызовов

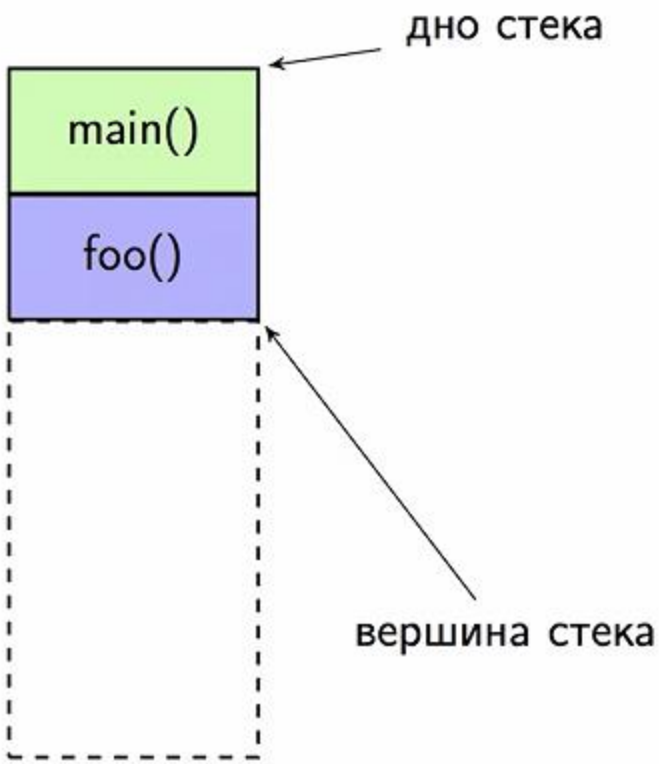
- Стек вызовов — это сегмент данных, используемый для хранения локальных переменных и временных значений.
- Не стоит путать стек с одноимённой структурой данных, у стека в C++ можно обратиться к произвольной ячейке.
- Стек выделяется при запуске программы.
- Стек обычно небольшой по размеру (4Мб).
- Функции хранят свои локальные переменные на стеке.
- При выходе из функции соответствующая область стека объявляется свободной.
- Промежуточные значения, возникающие при вычислении сложных выражений, также хранятся на стеке.

Устройство стека



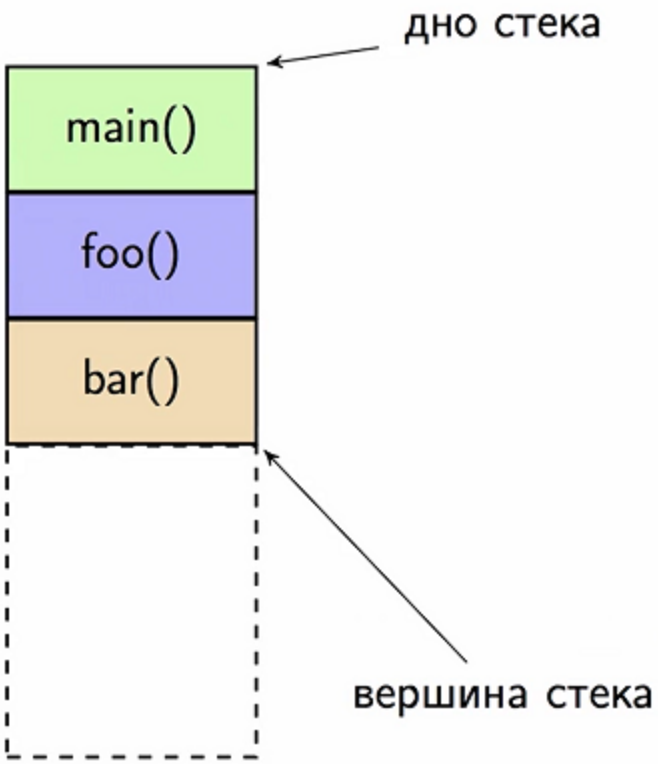

```
void bar( ) {
    int c;
}

void foo( ) {
    int b = 3;
    bar();
}

int main( ) {
    int a = 3;
    foo();
    bar();

    return 0;
}
```

Устройство стека

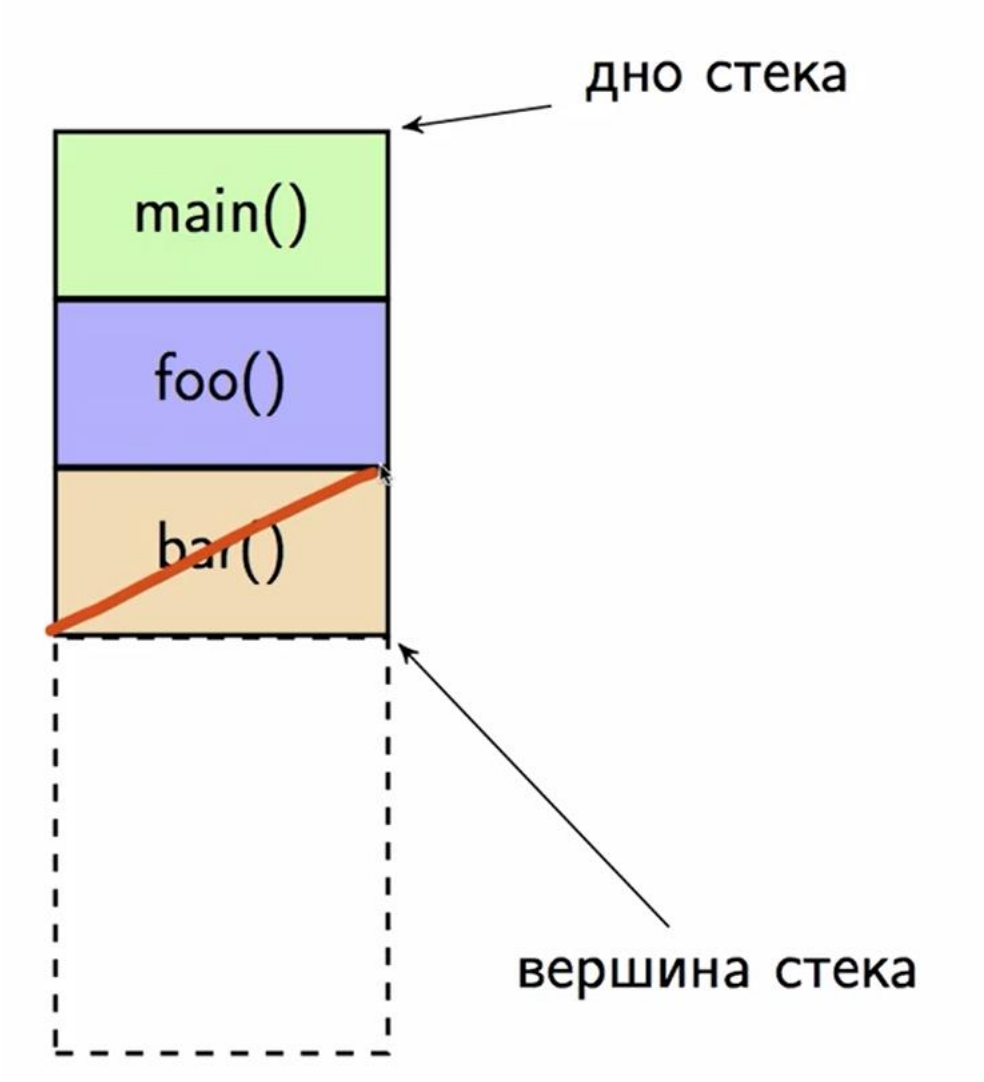


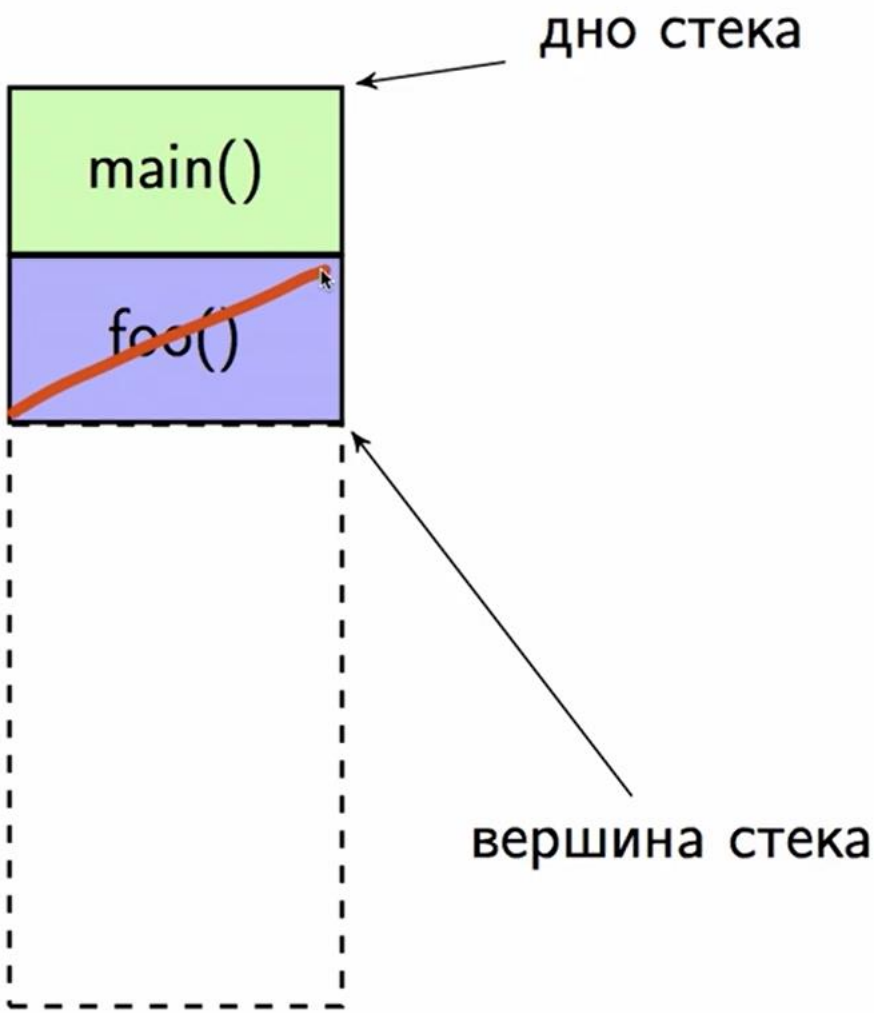

```
void bar( ) {
    int c;
}

void foo( ) {
    int b = 3;
    bar();
}

int main( ) {
    int a = 3;
    foo();
    bar();

    return 0;
}
```

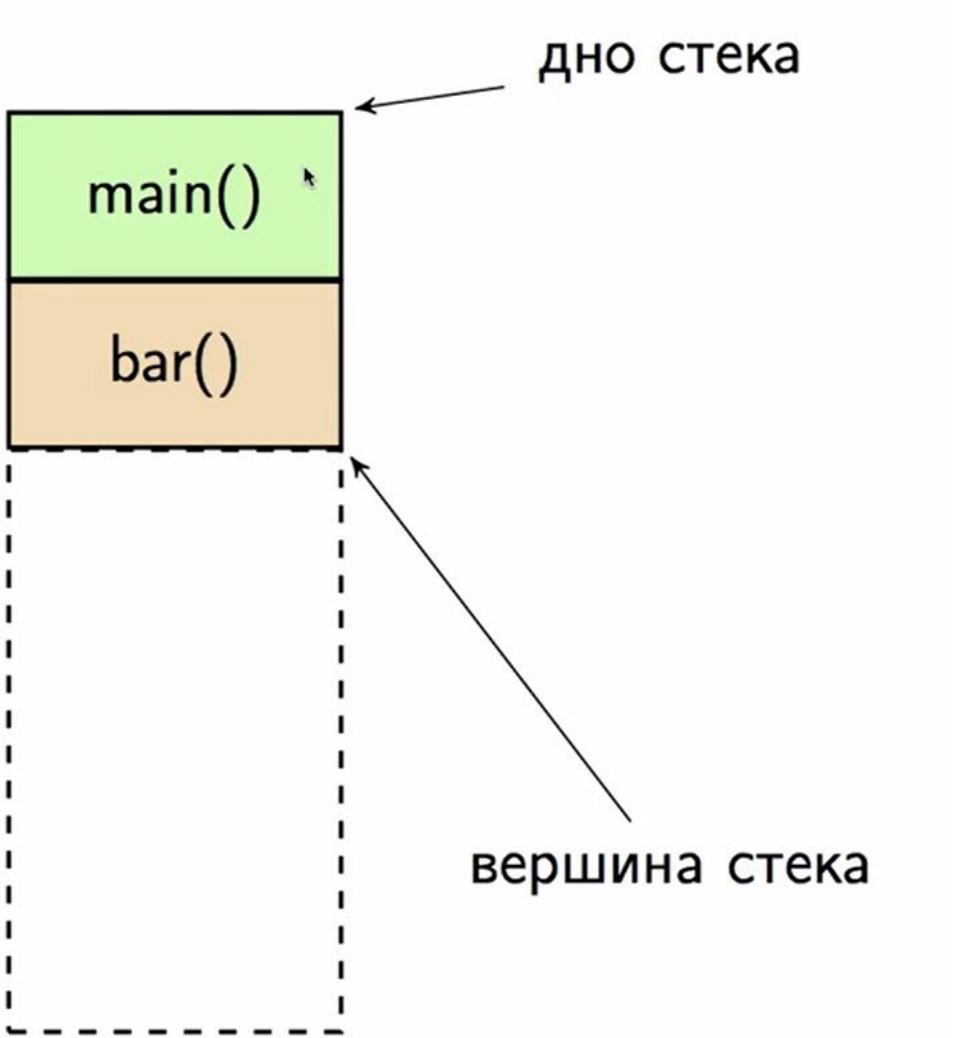
Устройство стека



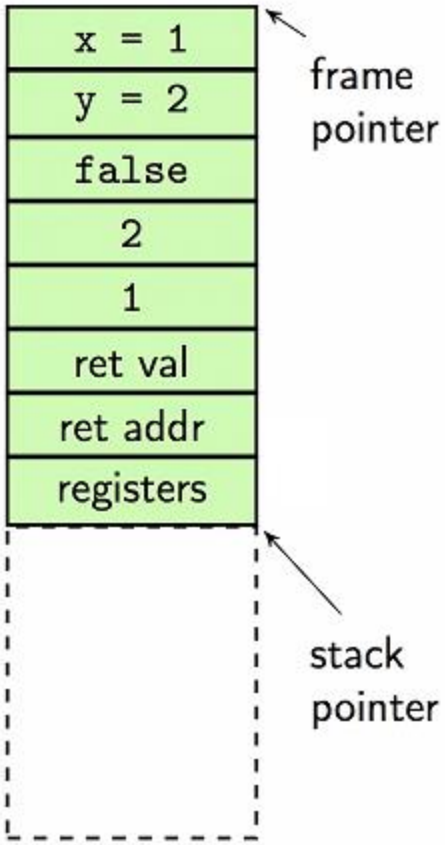

```
void bar( ) {
    int c;
}

void foo( ) {
    int b = 3;
    bar();
}

int main( ) {
    int a = 3;
    foo();
    bar();

    return 0;
}
```


дно стека
main()
foo()
bar()
вершина стека


дно стека
main()
foo()
вершина стека

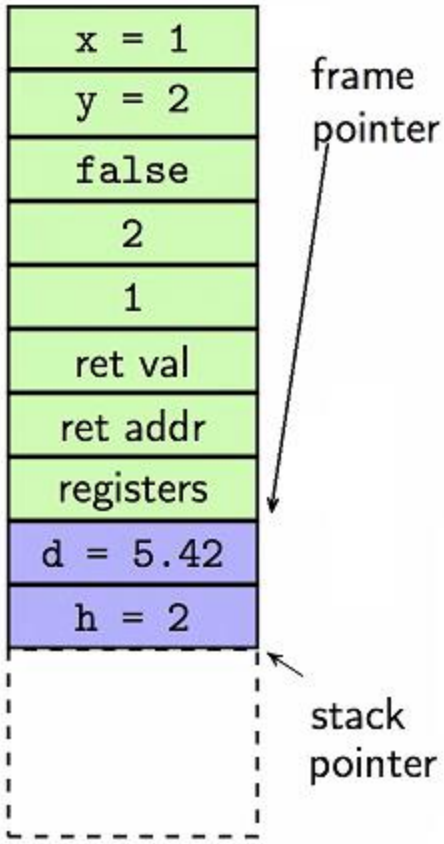

дно стека
main()
bar()
вершина стека

Вызов функции

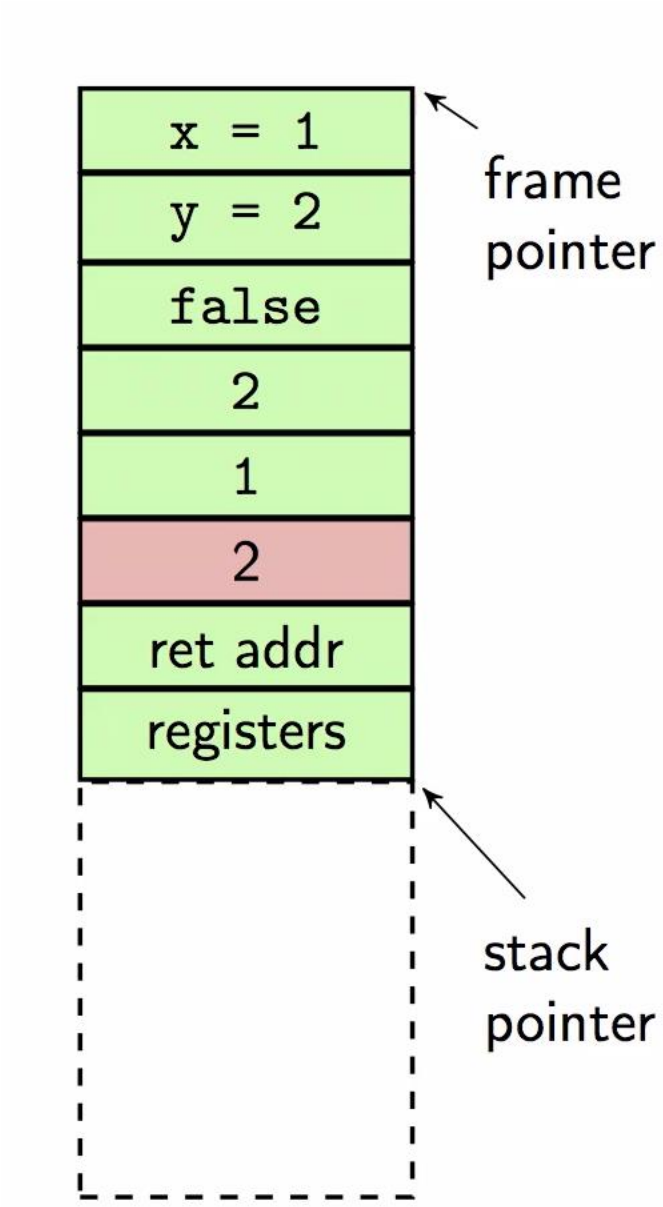


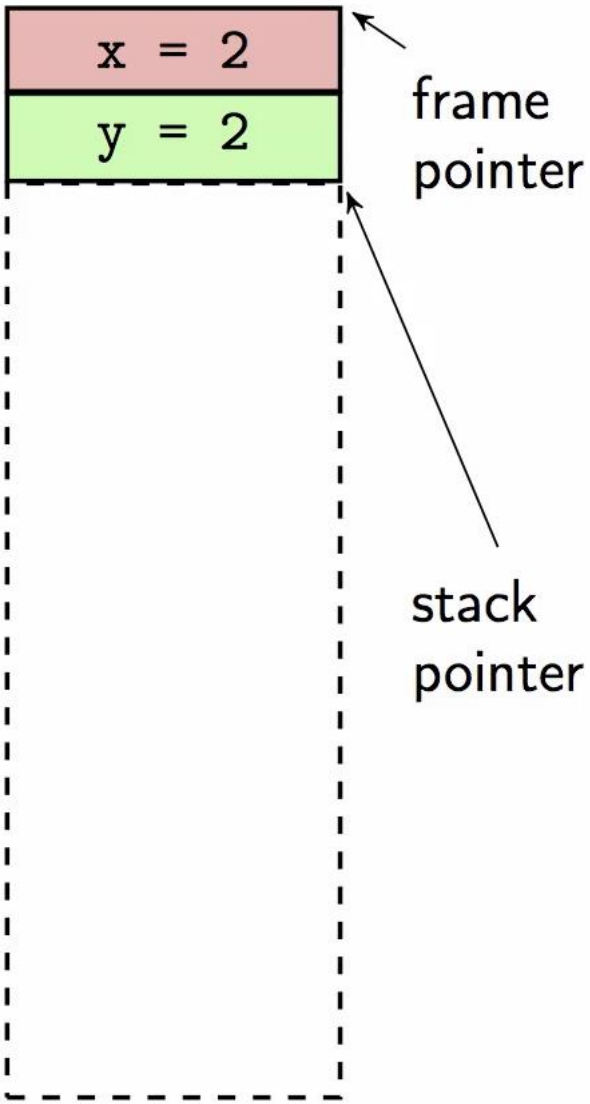

```
int foo(int a, int b, bool c)
{
    double d = a * b * 2.71;
    int    h = c ? d : d / 2;
    return h;
}

int main( )
{
    int x = 1;
    int y = 2;
    x = foo (x, y, false);
    cout << x;
    return 0;
}
```

Вызов функции




```
int foo(int a, int b, bool c)
{
    double d = a * b * 2.71;
    int    h = c ? d : d / 2;
    return h;
}

int main( )
{
    int x = 1;
    int y = 2;
    x = foo (x, y, false);
    cout << x;
    return 0;
}
```


x = 1
y = 2
false
2
1
2
ret addr
registers
frame pointer
stack pointer


x = 2
y = 2
frame
pointer
stack
pointer

Вызов функции

- При вызове функции на стек складываются:
  1. аргументы функции,
  2. адрес возврата,
  3. значение frame pointer и регистров процессора.
- Кроме этого на стеке резервируется место под возвращаемое значение.
- Параметры передаются в обратном порядке, что позволяет реализовать функции с переменным числом аргументов.
- Адресация локальных переменных функции и аргументов функции происходит относительно frame pointer.
- Конкретный процесс вызова зависит от используемых соглашений (cdecl, stdcall, fastcall, thiscall).

Уязвимость переполнения буфера

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

int foo() {
    cout << "Hello" << endl;
    return 2;
}

int bar() {
    int * m[1];
    m[3] = (int *) &foo;
    return 1;
}

int main() {
    bar();

    return 0;
}
```

Переопределение методов (overriding)

```
struct Person {
    string name() const { return name_; }
    ...
};
struct Professor : Person {
    string name() const {
        return "Prof. " + Person::name();
    }
    ...
};
```

```
    Professor pr("Stroustrup");
    cout << pr.name() << endl; // Prof. Stroustrup
    Person * p = &pr;
    cout << p->name() << endl; // Stroustrup
```

Решение проблемы:

```
struct Person {
    virtual string name() const { return name_; }
    ...
};
struct Professor : Person {
    string name() const {
        return "Prof. " + Person::name();
    }
    ...
};
```

Шаблонные функции

Поиск максимума

```
1 int max(int a, int b)
2 {
3     return (a > b) ? a : b;
4 }
```

```
1 double max(double a, double b)
2 {
3     return (a > b) ? a : b;
4 }
```

Шаблонный вариант

```
1 template <typename T> // объявление параметра шаблона функции
2 T max(T a, T b)
3 {
4     return (a > b) ? a : b;
5 }
```

Применение

```cpp
#include <iostream>

template <typename T>
const T& max(const T& a, const T& b)
{
    return (a > b) ? a : b;
}

int main()
{
    int i = max(4, 8);
    std::cout << i << '\n';

    double d = max(7.56, 21.434);
    std::cout << d << '\n';

    char ch = max('b', '9');
    std::cout << ch << '\n';

    return 0;
}
```