Основы Программирования

Лекция #2

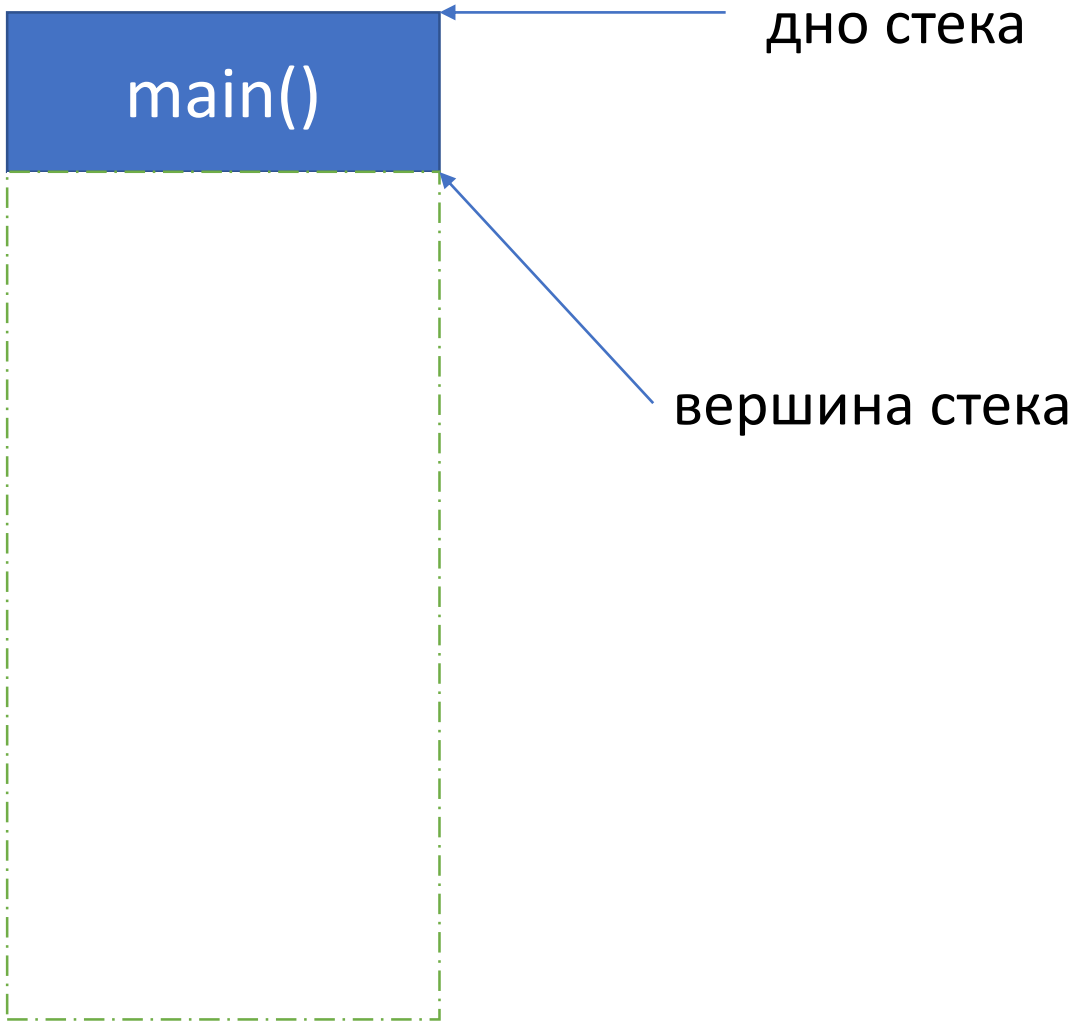
Темы:

1. Стек вызовов
2. Указатели и массивы
3. Строки
4. vector

Стек вызовов

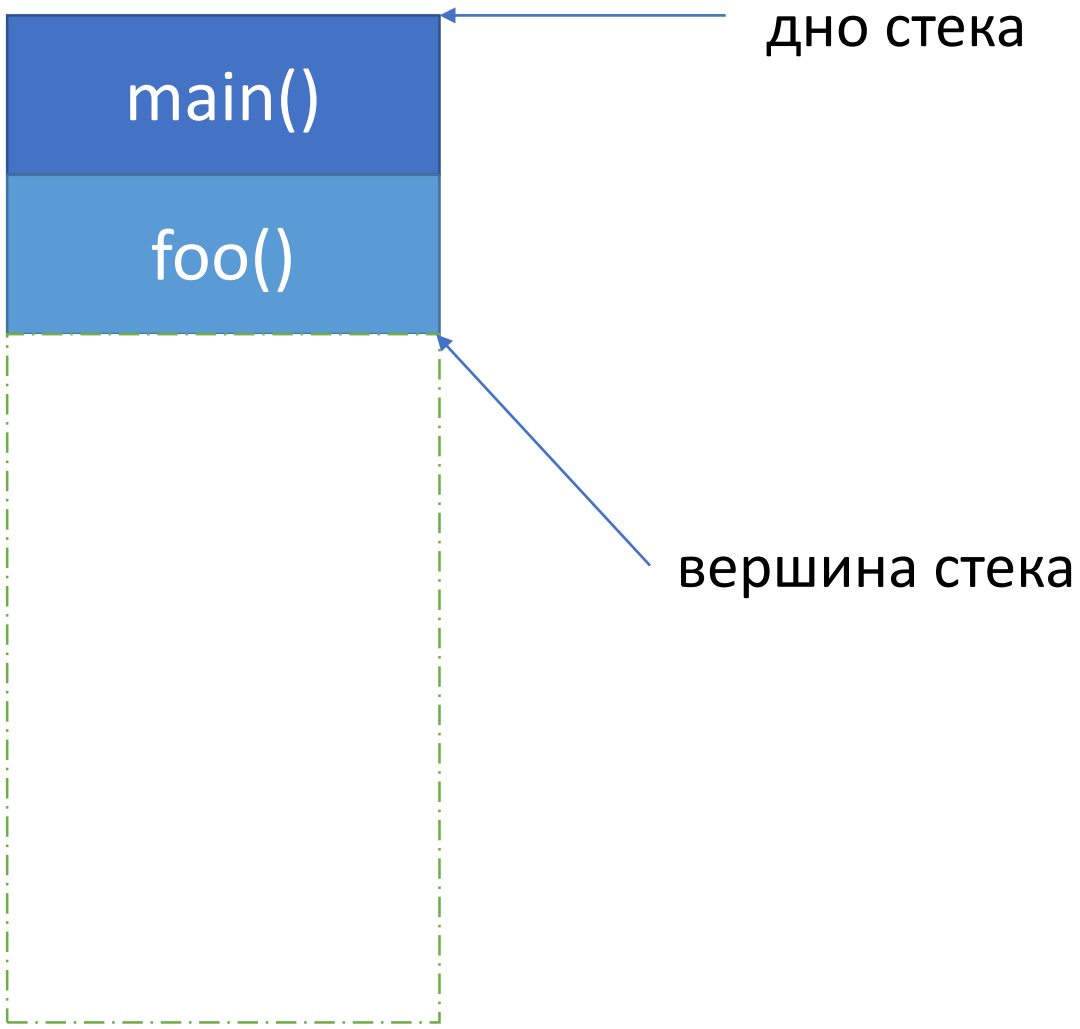
- Стек вызовов — это сегмент данных, используемый для хранения локальных переменных и временных значений.
- Не путать стек с одноимённой структурой данных, у стека в C++ можно обратиться к произвольной ячейке.
- Стек выделяется при запуске программы.
- Стек обычно небольшой по размеру (4Мб).
- Функции 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 1



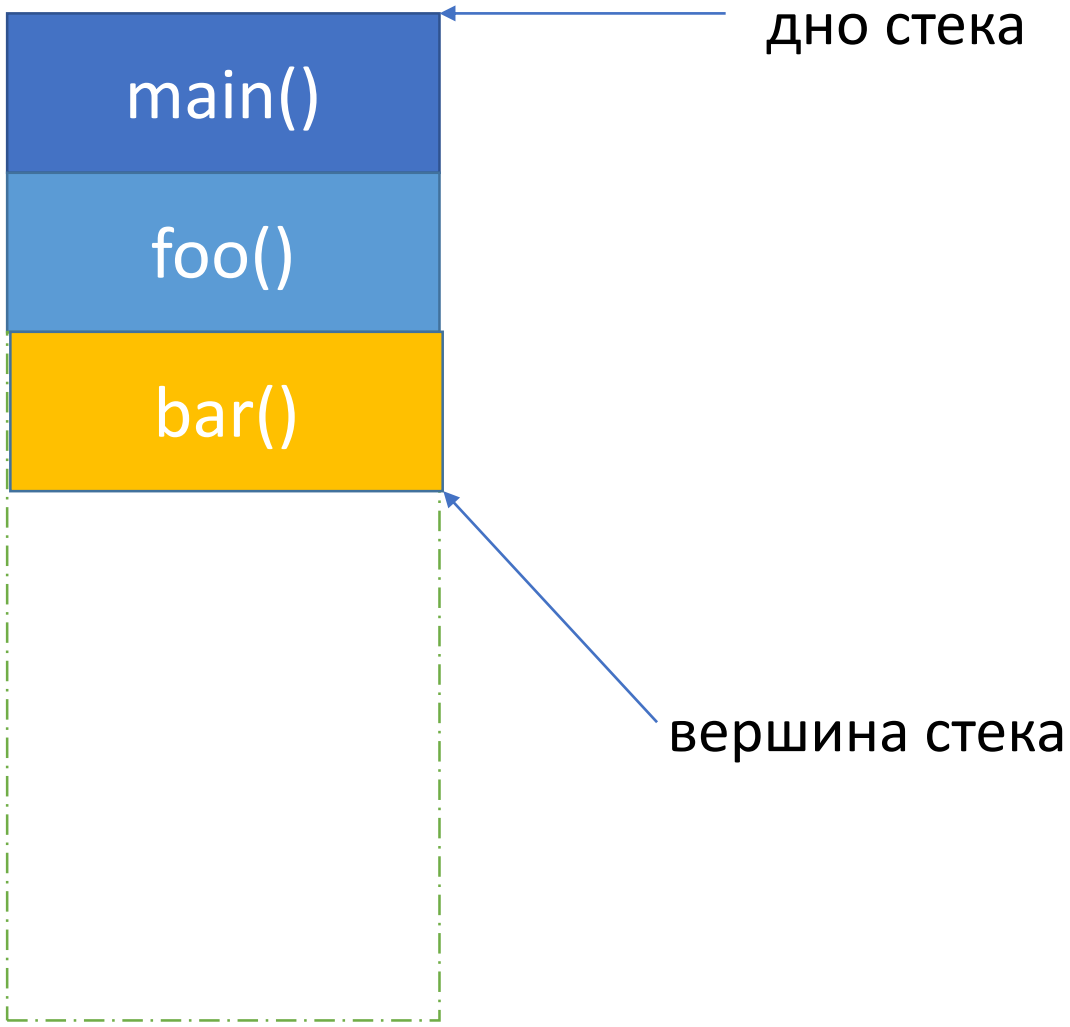

```
void bar () {
    int c;
}
void foo () {
    int b = 3;
    bar();
}
int main () {
    int a = 3;
    foo();
    bar();
    return 0;
}
```

Устройство стека 2



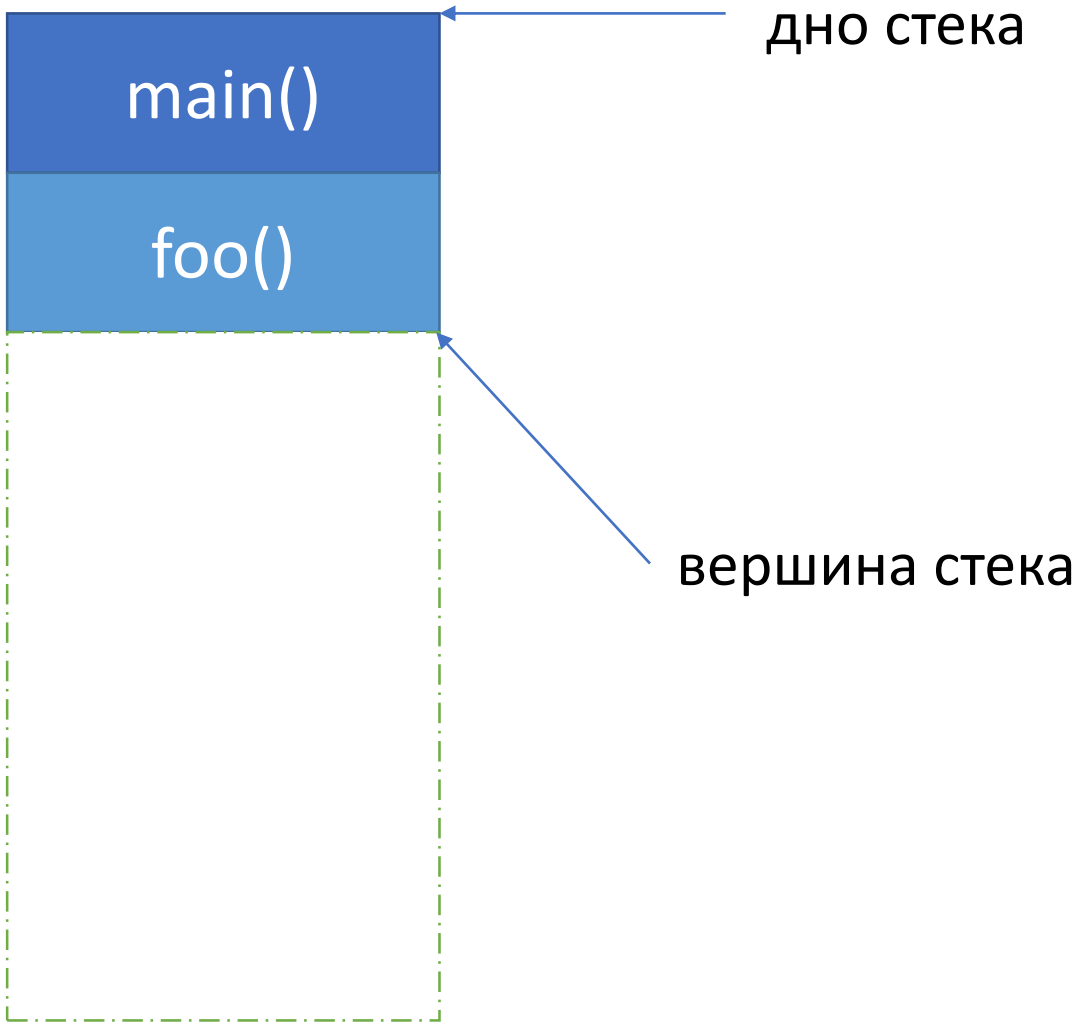

```
void bar () {
    int c;
}
void foo () {
    int b = 3;
    bar();
}
int main () {
    int a = 3;
    foo();
    bar();
    return 0;
}
```

Устройство стека 3



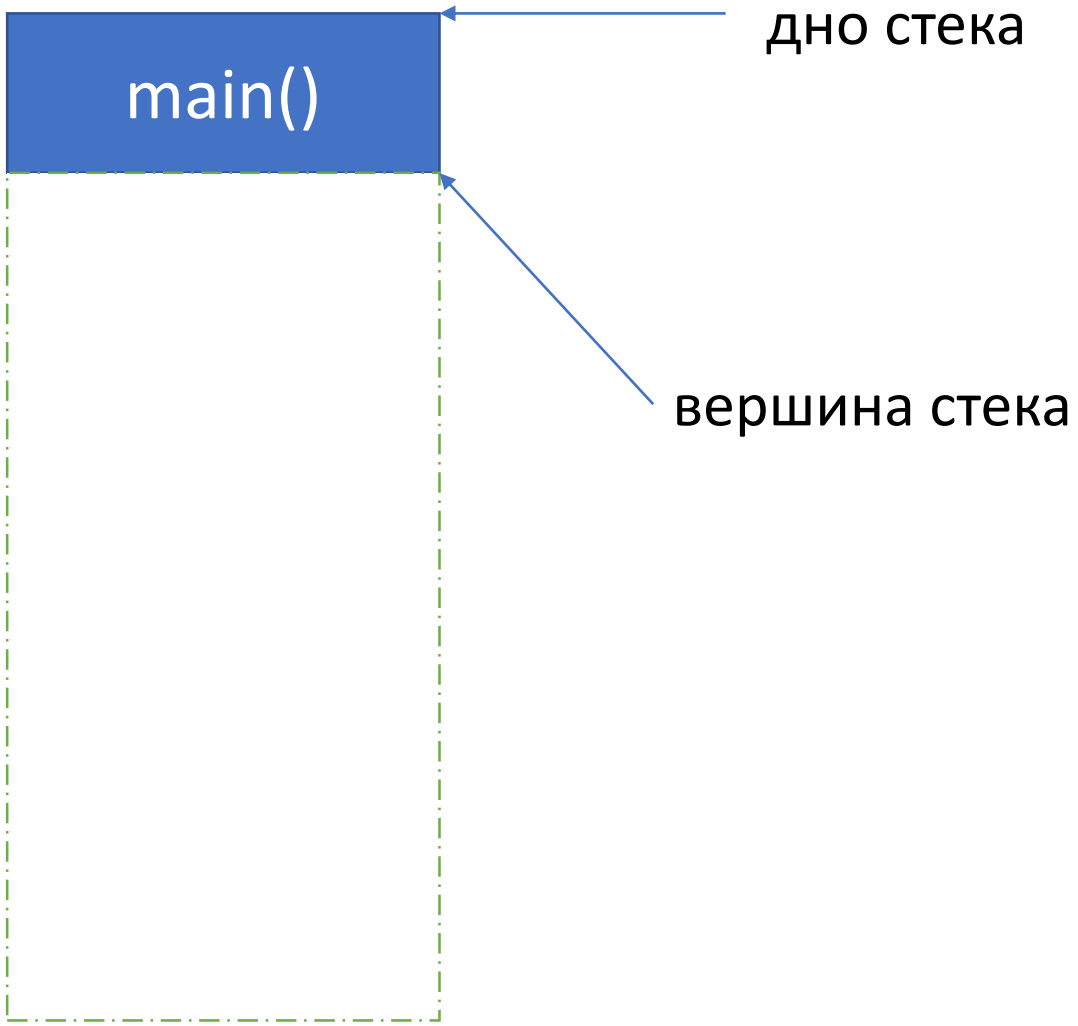

```
void bar () {
    int c;
}
void foo () {
    int b = 3;
    bar();
}
int main () {
    int a = 3;
    foo();
    bar();
    return 0;
}
```

Устройство стека 4



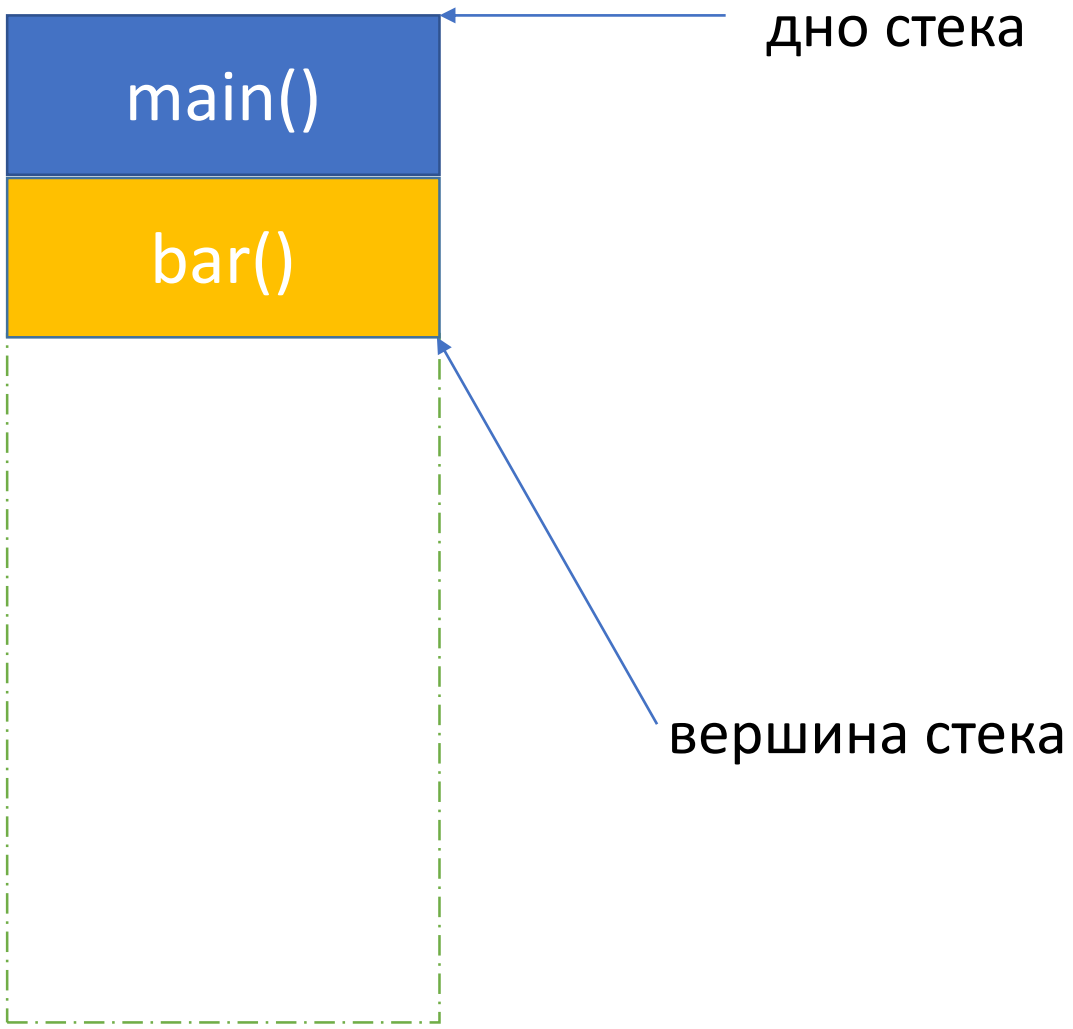

```
void bar () {
    int c;
}
void foo () {
    int b = 3;
    bar();
}
int main () {
    int a = 3;
    foo();
    bar();
    return 0;
}
```

Устройство стека 5



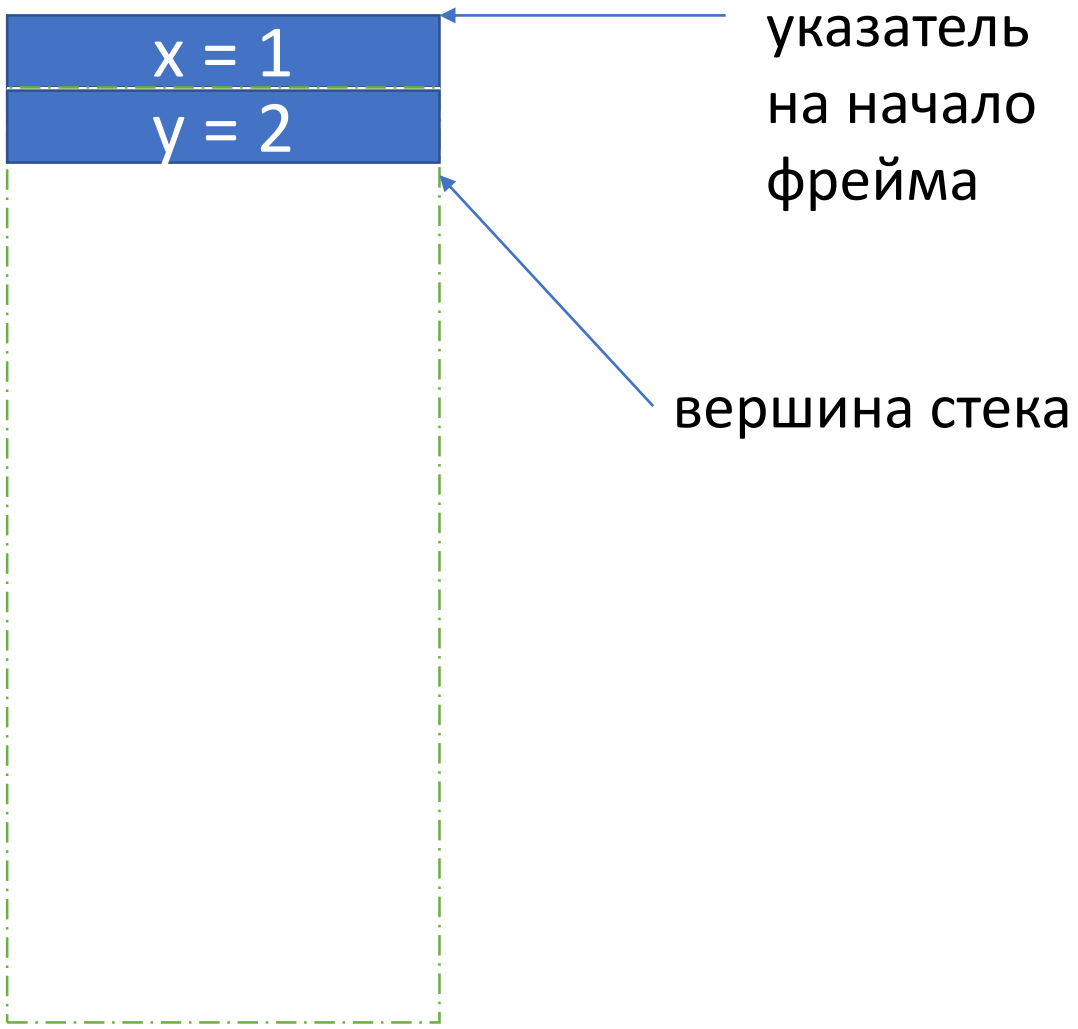

```
void bar () {
    int c;
}
void foo () {
    int b = 3;
    bar();
}
int main () {
    int a = 3;
    foo();
    bar();
    return 0;
}
```

Устройство стека 6




```
void bar () {
    int c; // чему равно значение?
}
void foo () {
    int b = 3;
    bar();
}
int main () {
    int a = 3;
    foo();
    bar();
    return 0;
}
```

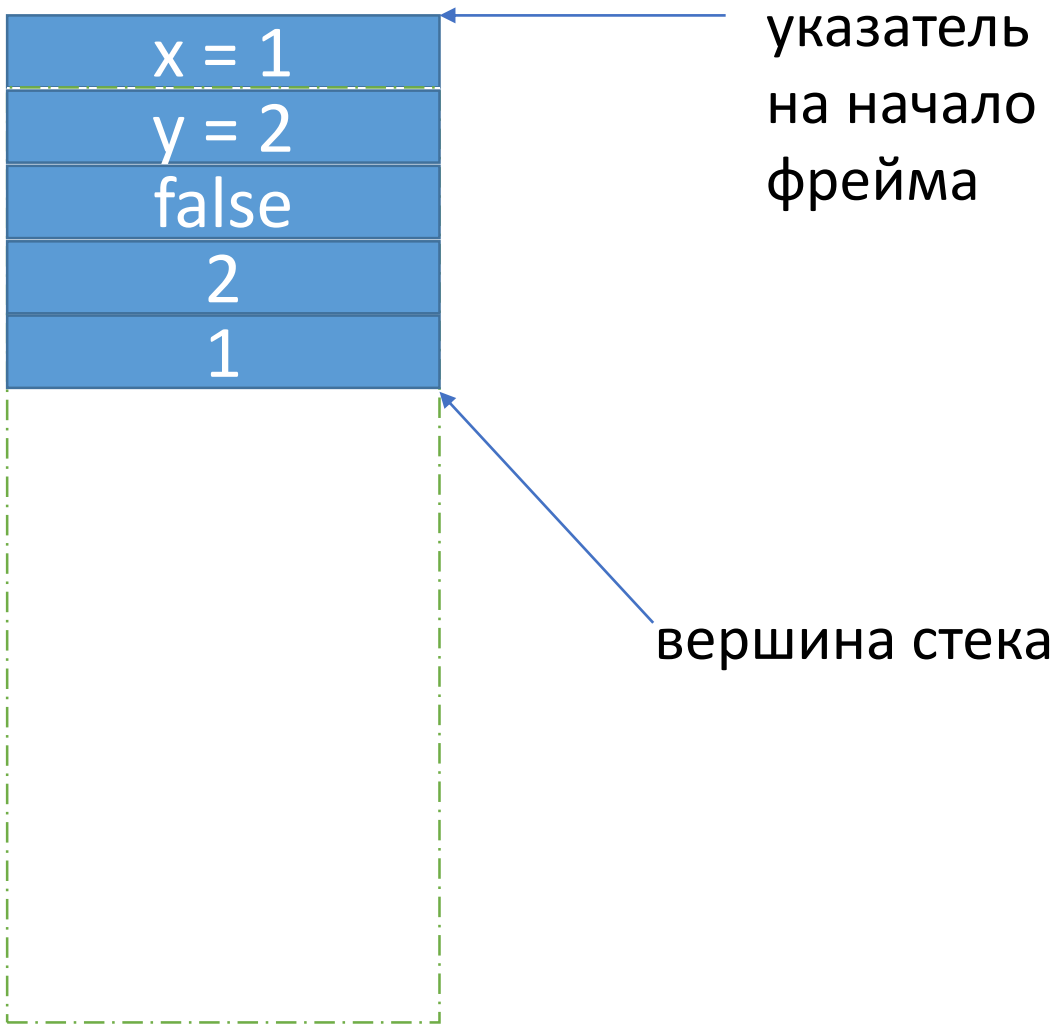
Вызов функции 1




```
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;
}
```

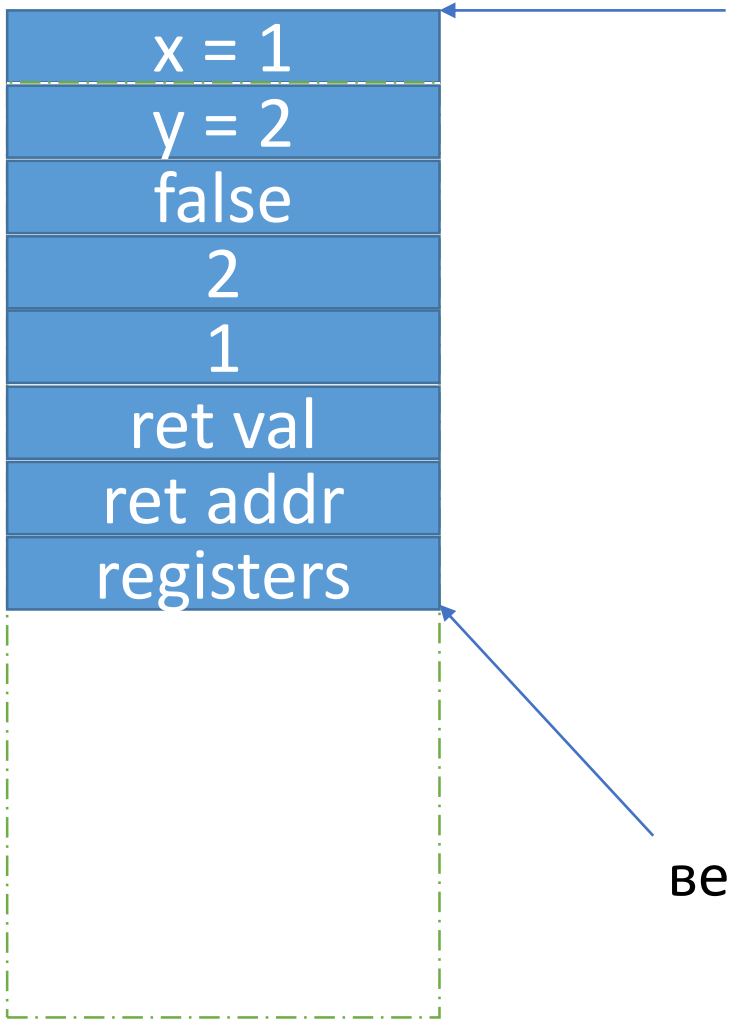
Вызов функции 2




```
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 3



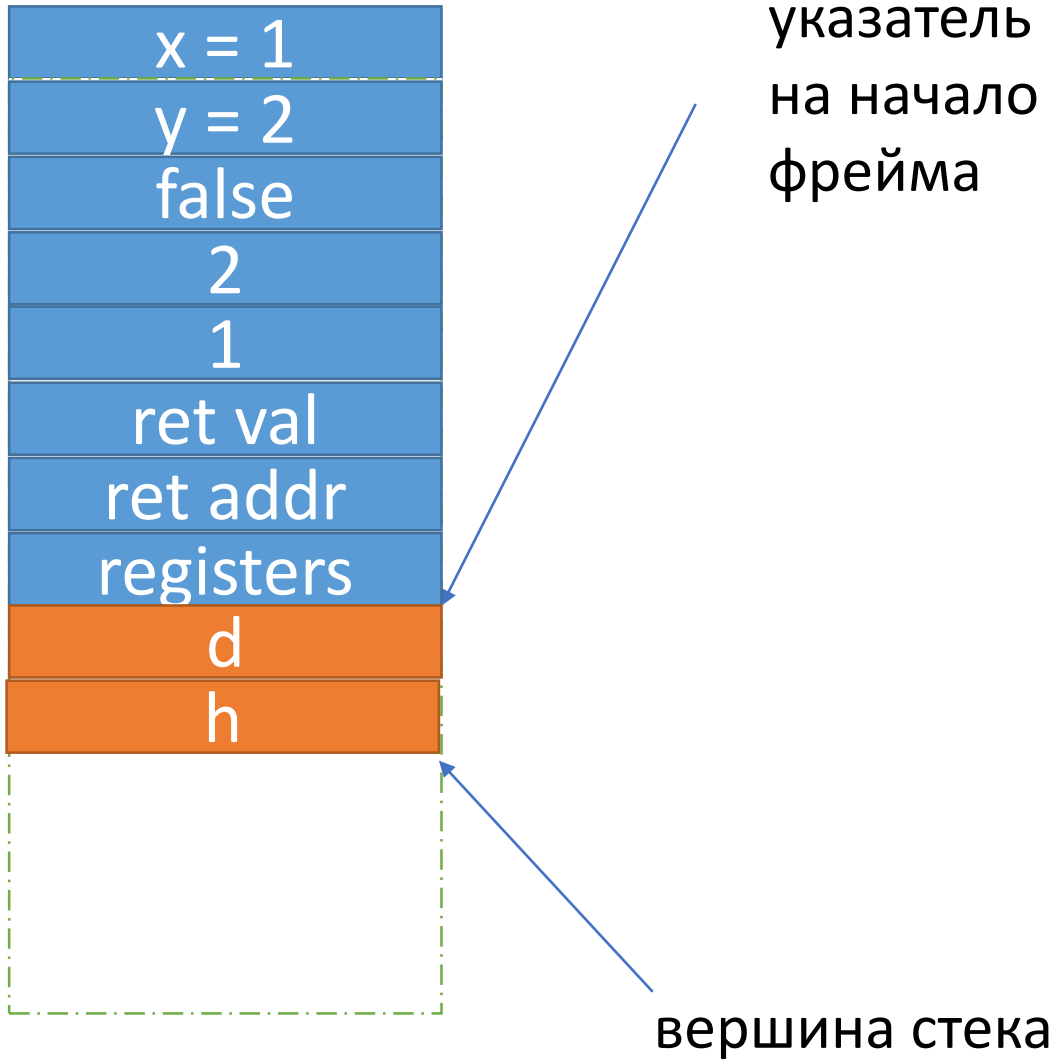
указатель на начало фрейма

вершина стека

```
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;
}
```

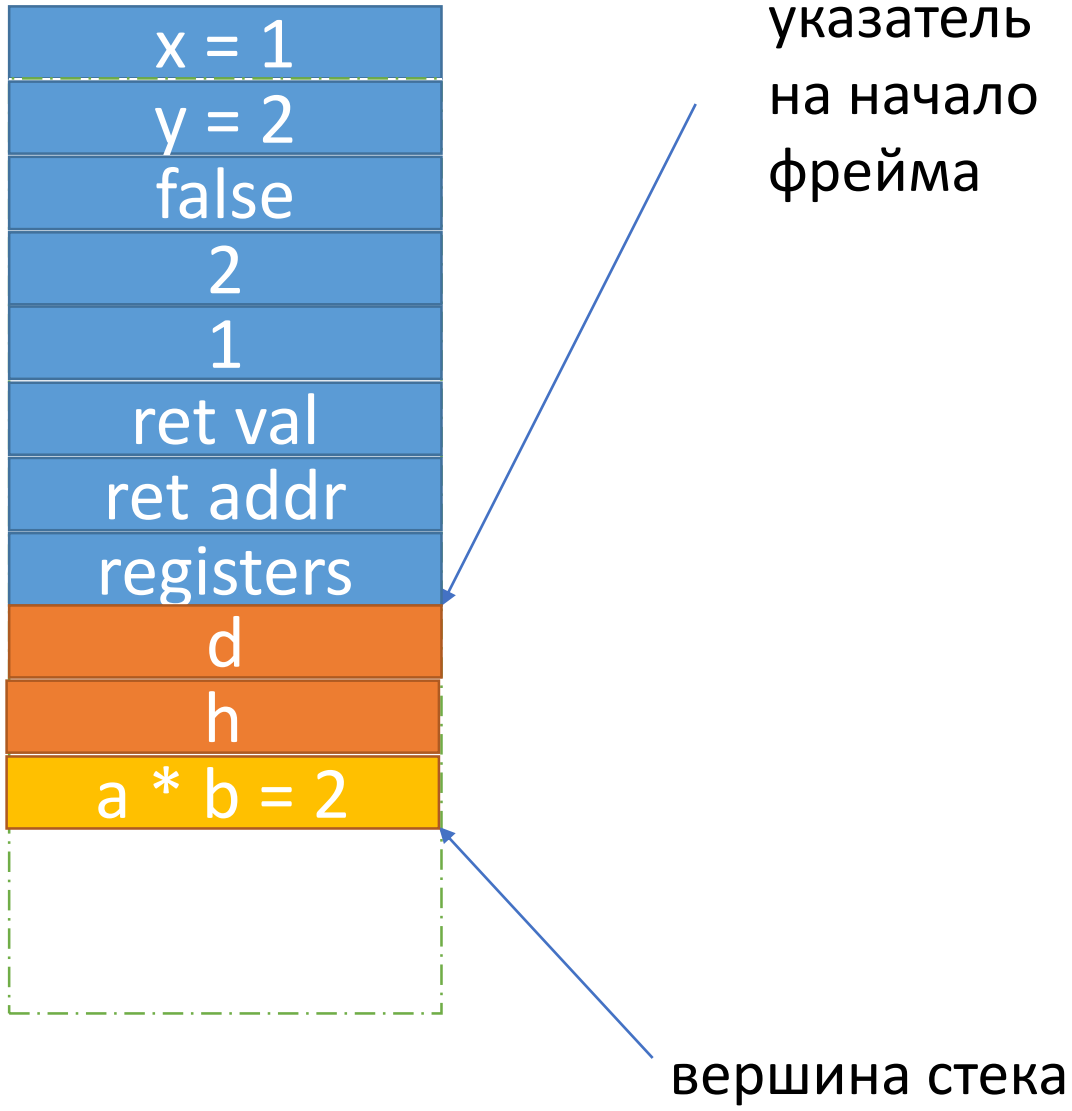
Вызов функции 4




```
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;
}
```

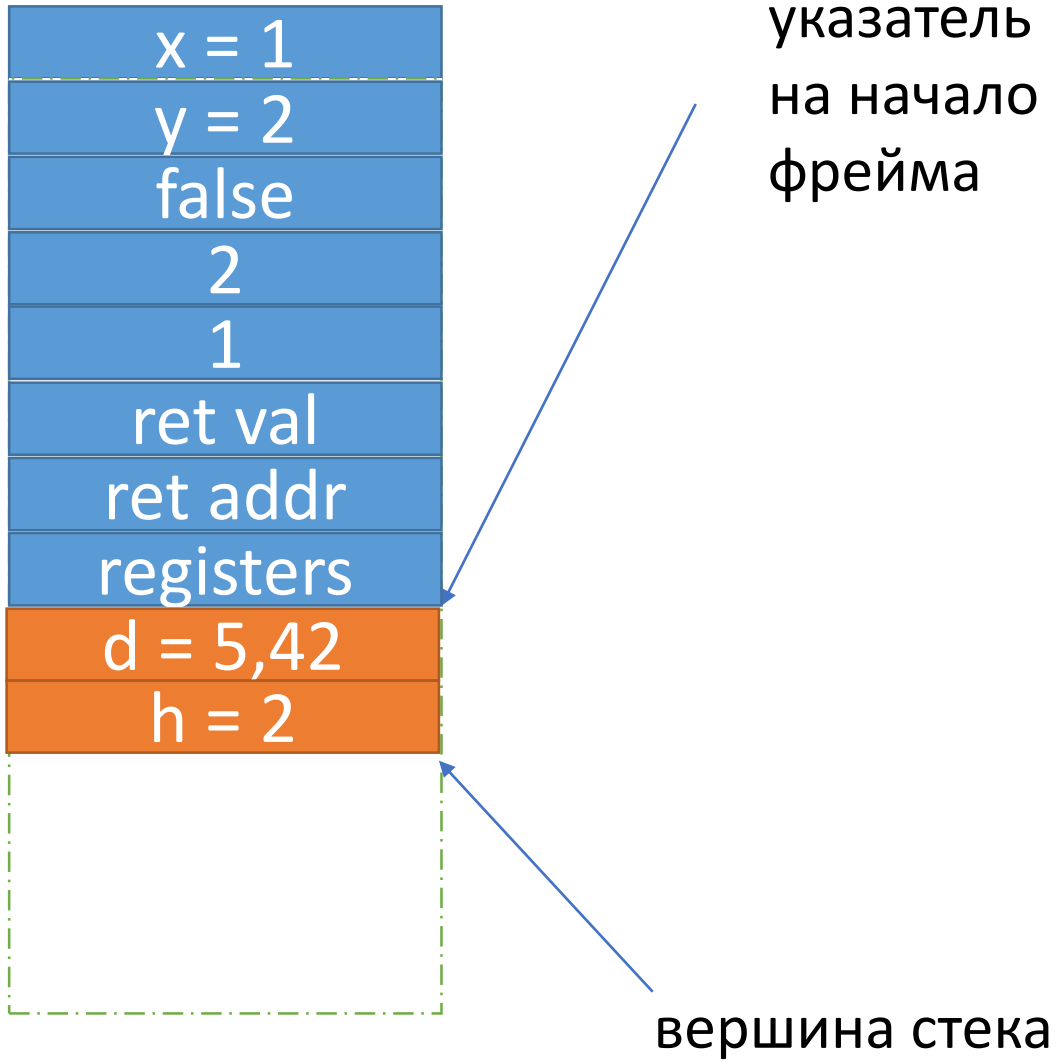
Вызов функции 5




```
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;
}
```

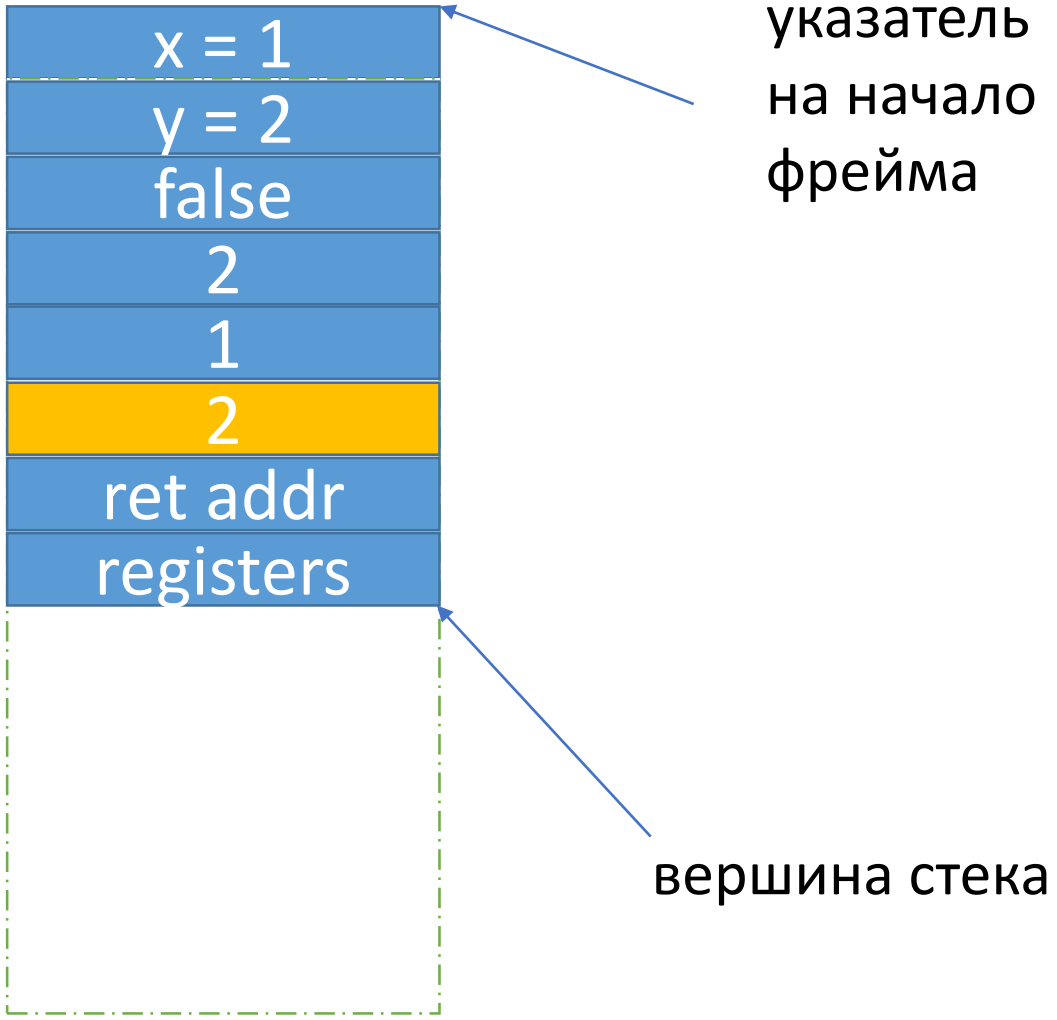
Вызов функции 6




```
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;
}
```

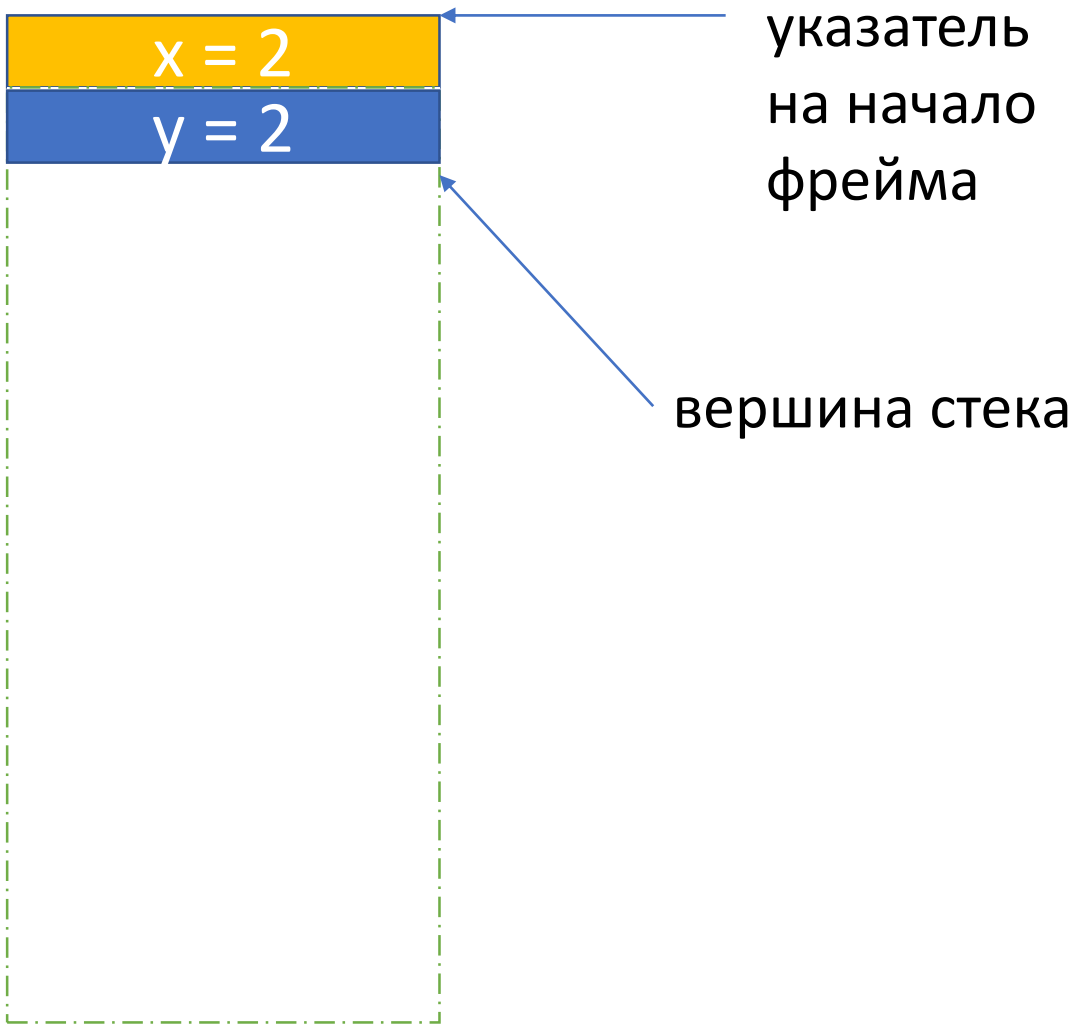
Вызов функции 7




```
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 8




```
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

- 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 указателя начала фрейма (frame pointer).

Указатели

- Указатель — это переменная, хранящая адрес некоторой ячейки памяти.
- Указатели являются типизированными.

```
int i = 3; // переменная типа int
int * p = 0; // указатель на переменную типа int
```

- Нулевому указателю (которому присвоено значение 0 или nullptr) не соответствует никакая ячейка памяти.
- Оператор взятия адреса переменной &.
- Оператор разыменования *.

```
p = &i; // указатель p указывает на переменную i
*p = 10; // изменяется ячейка по адресу p, т.е. i
```

Пример. Передача параметров по значению (копируются)

```
void swap(int a, int b)
{
	int t = a;
	a = b;
	b = t;
}
int main()
{
	int k = 10, m = 20;
	swap(k, m);
	cout << k << " " << m << endl; // 10 20
	return 0;
}
```

Пример. Передача параметров по указателю (без копии)

```
void swap(int *a, int *b)
{
        int t = *a;
        *a = *b;
        *b = t;
}
int main()
{
        int k = 10, m = 20;
        swap(&k, &m);
        cout << k << " " << m << endl; // 20 10
        return 0;
}
```

Массивы

- Массив — это набор однотипных элементов, расположенных в памяти друг за другом, доступ к которым осуществляется по индексу.

```
// массив 1 2 3 4 5 0 0 0 0 0
int m[10] = {1, 2, 3, 4, 5};
```

- Индексация массива начинается с 0, последний элемент массива длины n имеет индекс n - 1.

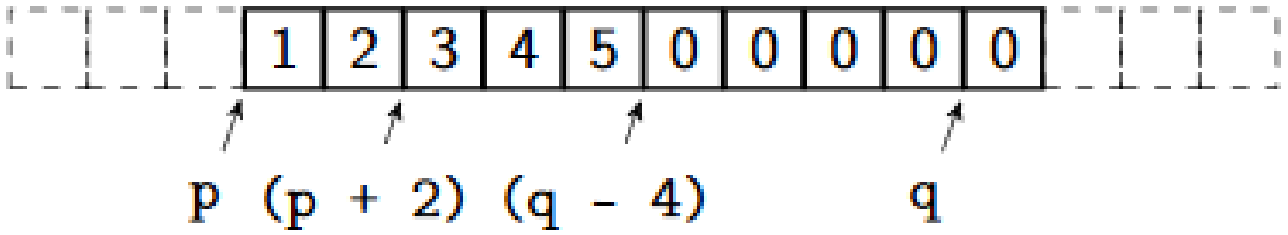
```
for (int i = 0; i < 10; i++)
      cout << m[i] << ' ';
cout << endl;
```

Связь массивов и указателей

- Указатели позволяют передвигаться по массивам.

```
int m[10] = {1, 2, 3, 4, 5};
int *p = &m[0]; // адрес начала массива
int *q = &m[9]; // адрес последнего элемента
```

- (p + k) — сдвиг на k ячеек типа int вправо.
- (p - k) — сдвиг на k ячеек типа int влево.
- (q - p) — количество ячеек между указателями.
- p[k] эквивалентно *(p + k)



Примеры

```
// Заполнение массива
int m[10] = {}; // изначально заполнен нулями
for (int *p = m; p <= m + 9; p++)
*p = (p - m) + 1;
// Массив заполнен числами от 1 до 10


// Передача массива в функцию
int max_element(int *m, int size)
{
        int max = *m;
        for (int i = 1; i < size; ++i)
                if (m[i] > max)
                        max = m[i];
        return max;
}
```

C-style строки

- В языке C строки представляли как массивы char-ов, которые заканчиваются специальным символом ‘\0’ (на самом деле, это просто символ с номером 0).
- C++ сохраняет (в основном) совместимость с языком C, и поэтому поддерживает работу с C-style строками.

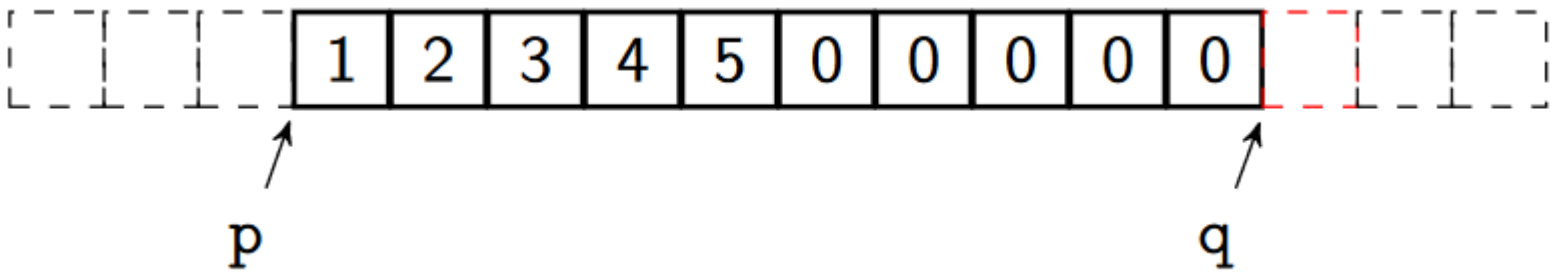
```
char s[15] = "C-style string";
char s[] = "C-style string"; // размер массива
15 char-ов
```

- При передаче в функцию массивы не копируются (т.е. на самом деле передаётся указатель на массив)

Поиск элемента в массиве

```
bool contains(int *m, int size, int value)
{
	for (int i = 0; i != size; i++)
		if (m[i] == value)
			return true;
	return false;
}

bool contains(int *p, int *q, int value)
{
	for (; p != q; p++)
		if (*p == value)
			return true;
	return false;
}
```



Функция поиска максимума в массиве

```
int max_element(int* p, int* q)
{
        int max = *p;
        for (; p != q; p++)
                if (*p > max)
                        max = *p;
        return max;
}

//....

int m[10] = {...};
int max = max_element(m, m + 10);
cout << "Maximum = " << max << endl;
```

Функция поиска максимума в массиве

```
int* max_element(int* p, int* q)
{
        int *pmax = p;
        for (; p != q; p++)
                if (*p > *pmax)
                        pmax = p;
        return pmax;
}

//....

int m[10] = {...};
int* pmax = max_element(m, m + 10);
cout << "Maximum = " << *pmax << endl;
```

Функция поиска максимума в массиве

```
int* max_element(int* p, int* q)
{
        int *pmax = p;
        for (; p != q; p++)
                if (*p > *pmax)
                        pmax = p;
        return pmax;
}

//....

int m[10] = {...};
int* pmax = max_element(m, m + 10);
cout << "Maximum = " << *pmax << endl;
```

Функция поиска максимума в массиве

```
bool max_element(int* p, int* q, int* res)
{
        if (p == q)
                return false;

        *res = *p;
        for (; p != q; p++)
                if (*p > *res)
                        *res = *p;
        return true;
}

// ....

int m[10] = {...};
int max = 0;
if (max_element(m, m + 10, &max))
        cout << "Maximum = " << max << endl;
```

Функция поиска максимума в массиве

```
bool max_element(int* p, int* q, int** res)
{
        if (p == q)
                return false;

        *res = p;
        for (; p != q; p++)
                if (*p > **res)
                        *res = p;
        return true;
}

// .....

int m[10] = {...};
int *pmax = 0;
if (max_element(m, m + 10, &pmax))
        cout << "Maximum = " << *pmax << endl;
```

Сортировка

```
#include <algorithm> //здесь объявлена функция sort
using namespace std;
int a[100] = {...};
sort(a, a + 100);

//....

int *minptr = min_element(a, a + 100);
int *maxptr = max_element(a, a + 100);
```

C++ строки

```
string str = "23,4,56";
stringstream ss(str);
char ch;
int a, b, c;
ss >> a >> ch >> b >> ch >> c; // a = 23, b = 4, c = 56
```

- Класс string
- #include <string>
- Методы:
  - size_t size()
  - bool empty()
  - char* c_str()
  - string substr(size_t pos = 0, size_t n = npos)
- Класс stringstream
- Для определения признака окончания потока использовать метод eof()

C++ строки (find)

- unsigned int find(pattern)

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
	int p;
	string a = "сеносеносеноиголкасеносеносеносено";
	p = a.find("иголка");
	cout << p << endl; // 12
	cout << a.find("иголка") << endl; // 12
}
```

C++ строки (find)

- size_type find(const basic_string&)

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
        int q;
        string a = "сеносеносеноиголкасеносеносеносено";
        q = a.find("Y");
        cout << q << endl; // -1
        cout << a.find("Y") << endl; // 4294967295
}
```

- size_t – платформенно зависимый alias
- size_type – контейнерно зависимый alias
- string::npos – самое большое возможное число соответствующего типа

Vector

- Динамический массив
- #include<vector>
- Итераторы:
  - begin()
  - end()
  - rbegin()
  - rend()

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

int main() {
	vector<int> g1;
	vector<int>::iterator i;
	vector<int>::reverse_iterator ir;

	for (int i = 1; i <= 5; i++)
		g1.push_back(i);
	cout << "Output of begin and end\t:\t";
	for (i = g1.begin(); i != g1.end(); ++i)
		cout << *i << '\t’;
	cout << endl << "Output of rbegin and rend\t:\t";
	for (ir = g1.rbegin(); ir != g1.rend(); ++ir)
		cout << '\t' << *ir;

	return 0;
}
```

Vector

- Методы:
  - size()
  - max_size()
  - capacity()
  - empty()

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

int main()
{
	vector<int> g1;

	for (int i = 1; i <= 5; i++)
		g1.push_back(i);

	cout << "Size : " << g1.size();
	cout << "\nCapacity : " << g1.capacity();
	cout << "\nMax_Size : " << g1.max_size();

	return 0;
}
```

```
Size : 5
Capacity : 8
Max_Size : 461168601842738790
```

Vector

- Методы:
  - [index]
  - at(index)
  - front()
  - back()

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;
int main()
{
	vector <int> g1;
	for (int i = 1; i <= 10; i++)
		g1.push_back(i * 10);
	cout << "Reference operator [g] : g1[2] = " << g1[2];
	cout << endl;
	cout << "at : g1.at(4) = " << g1.at(4);
	cout << endl;
	cout << "front() : g1.front() = " << g1.front();
	cout << endl;
	cout << "back() : g1.back() = " << g1.back();
	cout << endl;
	return 0;
}
```

Reference operator [g] : g1[2] = 30
at : g1.at(4) = 50
front() : g1.front() = 10
back() : g1.back() = 100

Vector

- Методы:
  - assign(size, value)
  - push_back(value)
  - pop_back()

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

int main() {
        vector<int> g;
        g.assign(5, 10); // 5 elements with value 10 each

        int sum = 0;
        g.push_back(10);
        g.push_back(20);
        g.push_back(30);
        while (!g.empty()) {
                sum += g.back();
        g.pop_back();
        }
        cout << "The sum of the elements of g is : " << sum << '\n';
        return 0;
}
```

Vector

- Методы:
  - insert(iterator, value)
  - insert(iterator, size, value)
  - insert(iterator, first, last)

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
        vector <int> g(3, 10);
        vector <int> :: iterator it;
        it = g.begin();
        it = g.insert(it, 20);
        g.insert(it, 2, 30);
        int gq [] = {50, 60, 70};
        g.insert(g.begin(), gq, gq + 3);
        cout << "g contains : ";
        for (it = g.begin(); it < g.end(); it++)
                cout << *it << '\t’;
        return 0;
}
```

Vector

- Методы:
  - insert(iterator, value)
  - insert(iterator, size, value)
  - insert(iterator, first, last)

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
        vector <int> g(3, 10); // 10 10 10
        vector <int> :: iterator it;
        it = g.begin();
        it = g.insert(it, 20); // 20 10 10 10
        g.insert(it, 2, 30);  // 30 30 20 10 10 10
        int gq [] = {50, 60, 70};
        g.insert(g.begin(), gq, gq + 3); // 50 60 70 30...
        cout << "g contains : ";
        for (it = g.begin(); it < g.end(); it++)
                cout << *it << '\t’;
        return 0;
}
```

Vector

- Методы:
  - erase(iterator)
  - erase(first, last)

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

int main() {
	vector<int> g;

	for (int i = 1; i <= 10; i++)
		g.push_back(i * 2);
	// erase the 5th element
	g.erase(g.begin() + 4);
	// erase the first 5 elements:
	g.erase(g.begin(), g.begin() + 5);
	cout << "g contains :";
	for (int i = 0; i < g.size(); ++i)
		cout << g[i] << '\t’;
	return 0;
}
```

Vector

- Методы:
  - swap(vector, vector)
  - clear()

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

int main() {
	vector<int> g1;
	vector<int> g2;
	g1.push_back(10);
	g1.push_back(20);
	g2.push_back(30);
	g2.push_back(40);

	swap(g1, g2);

	g1.clear();
	g1.push_back(1000);
	cout << g1.front();

return 0;
}
```