

Функции

гл. ас. д-р. Нора Ангелова

Функции

Дефиниция на функция

- Заглавие

<тип_на_функция> <име_на_функция> (<формални_параметри>)

<тип_на_функция> – име на тип.

Типът на резултата от изпълнението на функцията.

Ако функцията не връща резултат – тип void.

<име_на_функция> – идентификатор.

<формални_параметри> – множество от параметри. Те изпълняват ролята на входните данни на функцията.

Функцията може да няма формални параметри.

<параметър> – <име_на_тип> <име_на_параметър>.

- Тяло – редица от оператори и дефиниции оградени в { }.

{ <тяло> }

Функции

Декларация на функция

```
<тип_на_функция> <име_на_функция>(<формални_параметри>);
```

- Декларацията на функцията се нарича също **прототип**.
- Имената на формалните параметри в декларацията могат да се пропускат.

Пример:

```
void testFunction(int);  
void testFunction2(int a);
```

Функции

Извикване на функция

```
<име_на_функция>( <фактически_параметри> );
```

Пример:

```
testFunction(5);
```

Функции

- Преди да се извика една функция, тя трябва да е декларирана/дефинирана.

Ако дадена функция `example` извиква функцията `testFunction`, функцията `testFunction` трябва да бъде декларирана/дефинирана преди `example`.

ФУНКЦИИ

Пример:

```
void printParam(int a) {  
    cout << a << endl;  
}
```

```
int main() {  
    printParam(5);  
  
    return 0;  
}
```

ФУНКЦИИ

Пример:

```
void printParam(int a);
```

```
int main() {  
    printParam(5);  
  
    return 0;  
}
```

```
void printParam(int a) {  
    cout << a;  
}
```

ФУНКЦИИ

Пример:

```
void printParam(int);
```

```
int main() {  
    printParam(5);
```

```
    return 0;  
}
```

```
void printParam(int a) {  
    cout << a;  
}
```


Функции

Свързване на формални с фактически параметри:

- Формален параметър – стойност.
- Формален параметър – указател.

ФУНКЦИИ

```
void testFunction(int param) {  
    ...  
}  
    ...
```

param



```
int value = 5;  
testFunction(value);
```

value



addr1

ФУНКЦИИ

```
void testFunction(int *param) {  
    ...  
}  
    ...
```

param



```
int value = 5;  
testFunction(&value);
```

value



addr1

Функции

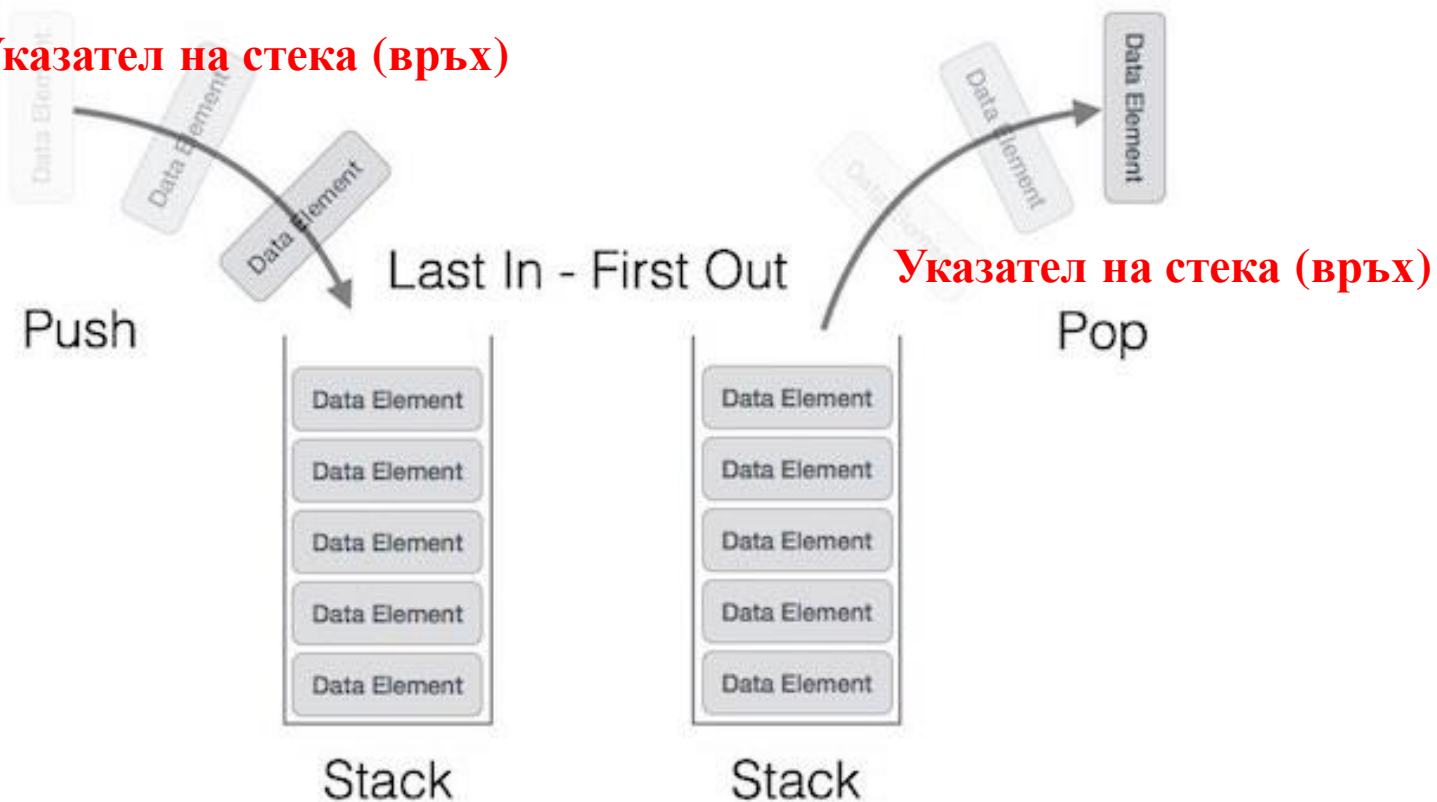
Разпределение на оперативната памет за програма:

- Програмен код – записан е изпълнимият код на всички функции, изграждащи програмата.
- Статични данни – записани са глобални и статични обекти.
- Динамични данни – чрез средства за динамично разпределение на паметта се заделя и освобождава памет в процеса на изпълнение на програмата (не преди това). Тази памет е от областта на динамичните данни.
- Програмен стек – записани са данните за функциите.

Стек

- Хомогенна линейна структура от елементи.
- „последен влязъл - пръв излязъл“ (LIFO).
- Пряк достъп, включване, изключване на елемент.

Указател на стека (връх)



Функции

Програмен стек

- Елементите на стека са „блокове“ от памет.
- Записани са данните за функциите.
- Наричат се още **стекови рамки**.

Функции

Програмен стек

- В дъното на стека е стековата рамка на `main`.
- На върха на стека е стековата рамка на функцията, която се обработва в момента.
- Под нея е стековата рамка на функцията, извикала функцията, обработваща се в момента.

Функции

Свързване на формални с фактически параметри:

- Формален параметър – стойност

В стековата рамка на функцията:

За формалния параметър се отделя толкова памет, колкото типът му изисква и в нея се копира стойността на фактическия параметър.

- Формален параметър – указател

В стековата рамка на функцията:

За формалния параметър се отделя памет за един указател. В нея се записва стойността на фактическия параметър, който трябва да бъде адрес на променлива.

Функции

```
int gcd(int x, int y) {
    while(x != y) {
        if (x > y) {
            x = x - y;
        } else {
            y = y - x;
        }
    }

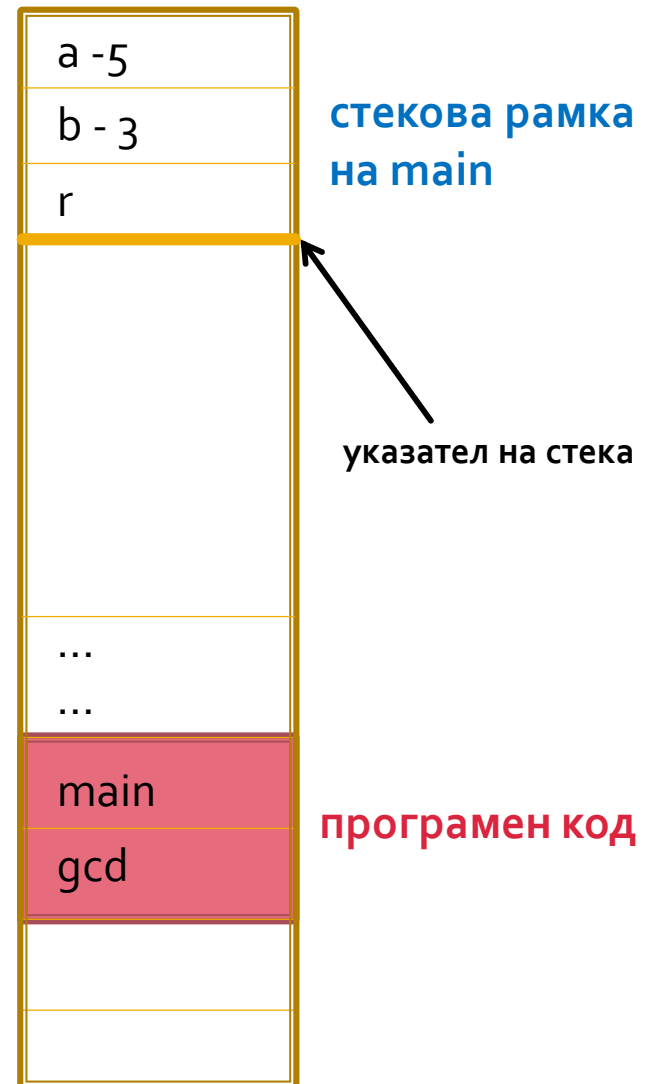
    return x;
}

int main() {
    int a = 5, b = 3;
    int r = gcd(a, b);

    cout << r << endl;
    return 0;
}
```

Функции

```
int gcd(int x, int y) {  
    while(x != y) {  
        if (x > y) {  
            x = x - y;  
        } else {  
            y = y - x;  
        }  
    }  
  
    return x;  
}  
  
int main() {  
    int a = 5, b = 3;  
    int r = gcd(a, b);  
  
    cout << r << endl;  
    return 0;  
}
```



Функции

```
int gcd(int x, int y) {  
    while(x != y) {  
        if (x > y) {  
            x = x - y;  
        } else {  
            y = y - x;  
        }  
    }  
  
    return x;  
}  
  
int main() {  
    int a = 5, b = 3;  
    int r = gcd(a, b);  
  
    cout << r << endl;  
    return 0;  
}
```



Функции

```
int gcd(int x, int y) {
    while(x != y) {
        if (x > y) {
            x = x - y;
        } else {
            y = y - x;
        }
    }

    return x;
}

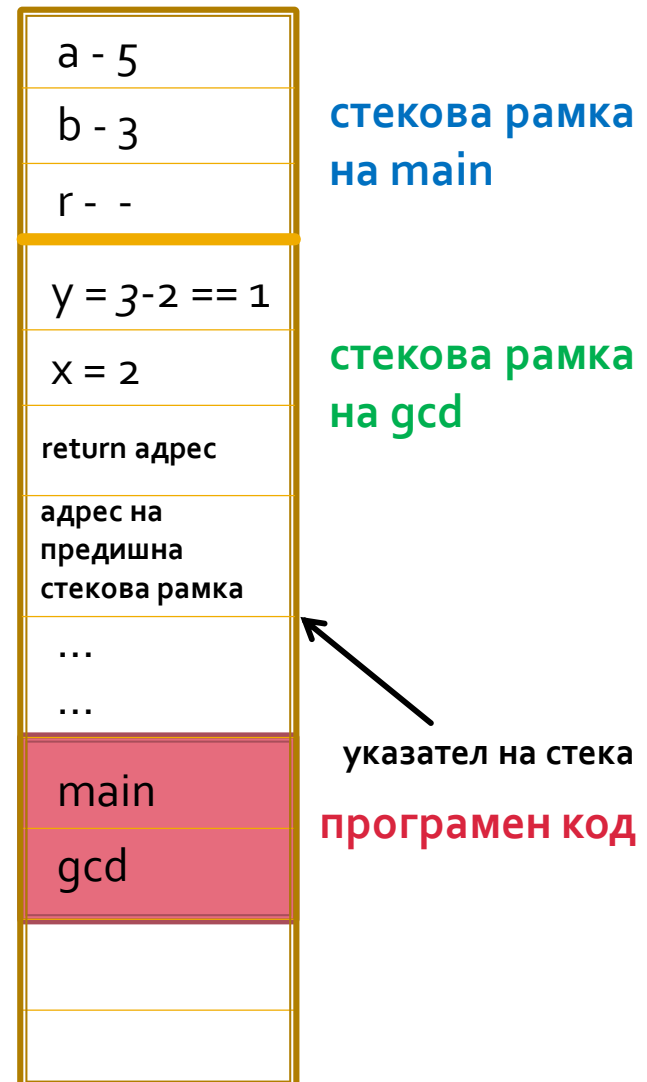
int main() {
    int a = 5, b = 3;
    int r = gcd(a, b);

    cout << r << endl;
    return 0;
}
```



Функции

```
int gcd(int x, int y) {  
    while(x != y) {  
        if (x > y) {  
            x = x - y;  
        } else {  
            y = y - x;  
        }  
    }  
  
    return x;  
}  
  
int main() {  
    int a = 5, b = 3;  
    int r = gcd(a, b);  
  
    cout << r << endl;  
    return 0;  
}
```



Функции

```
int gcd(int x, int y) {  
    while(x != y) {  
        if (x > y) {  
            x = x - y;  
        } else {  
            y = y - x;  
        }  
    }  
  
    return x;  
}  
  
int main() {  
    int a = 5, b = 3;  
    int r = gcd(a, b);  
  
    cout << r << endl;  
    return 0;  
}
```



Функции

```
int gcd(int x, int y) {
    while(x != y) {
        if (x > y) {
            x = x - y;
        } else {
            y = y - x;
        }
    }

    return x;
}

int main() {
    int a = 5, b = 3;
    int r = gcd(a, b);

    cout << r << endl;
    return 0;
}
```



Функции

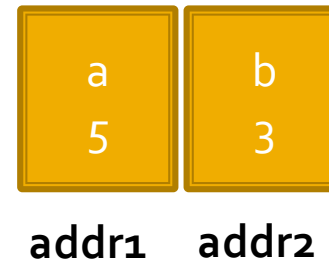
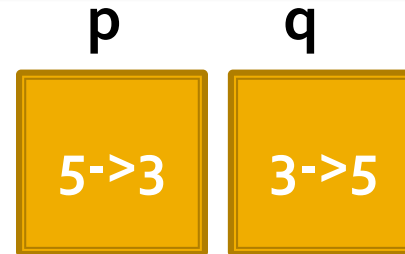
```
int gcd(int x, int y) {  
    while(x != y) {  
        if (x > y) {  
            x = x - y;  
        } else {  
            y = y - x;  
        }  
    }  
  
    return x;  
}  
  
int main() {  
    int a = 5, b = 3;  
    int r = gcd(a, b);  
  
    cout << r << endl;  
    return 0;  
}
```



Функции

```
void swap(int p, int q) {  
    int temp = p;  
    p = q;  
    q = temp;  
}
```

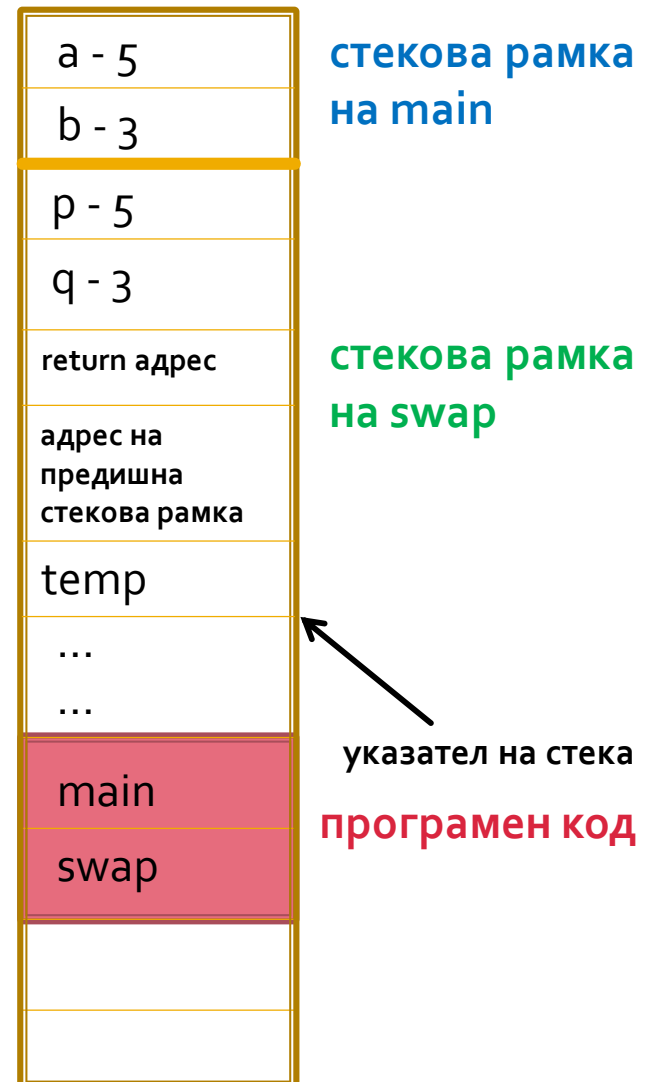
```
int main() {  
    int a = 5, b = 3;  
    swap(a, b); // 5 3  
    return 0;  
}
```



Функции

```
void swap(int p, int q) {  
    int temp = p;  
    p = q;  
    q = temp;  
}
```

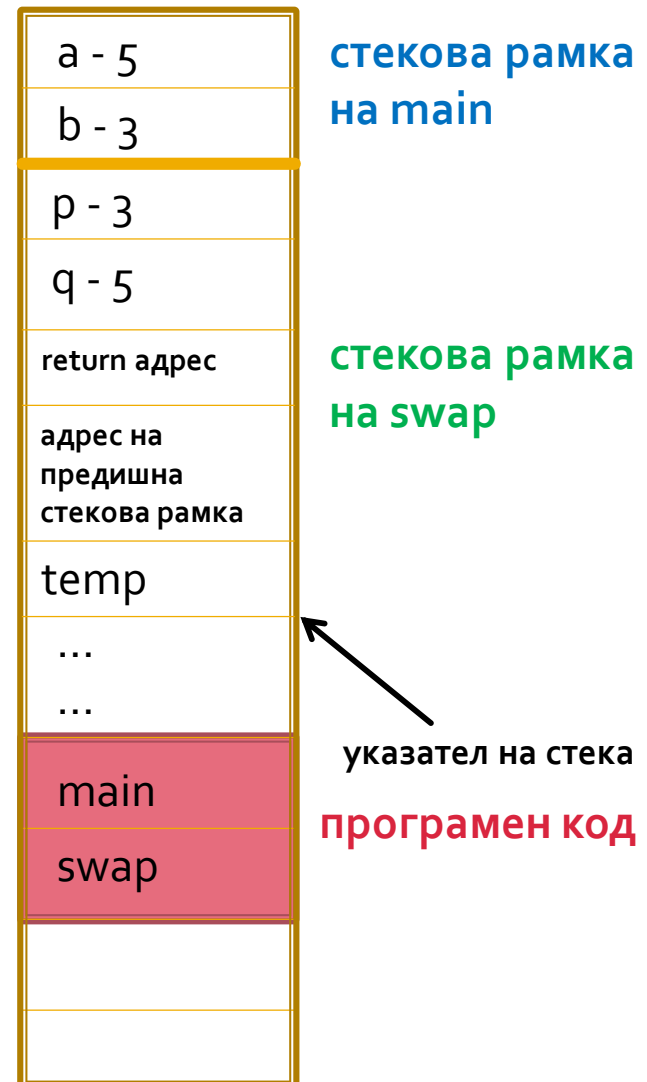
```
int main() {  
    int a = 5, b = 3;  
    swap(a, b); // 5 3  
    return 0;  
}
```



Функции

```
void swap(int p, int q) {  
    int temp = p;  
    p = q;  
    q = temp;  
}
```

```
int main() {  
    int a = 5, b = 3;  
    swap(a, b); // 5 3  
    return 0;  
}
```



Функции

```
void swap(int p, int q) {  
    int temp = p;  
    p = q;  
    q = temp;  
}
```

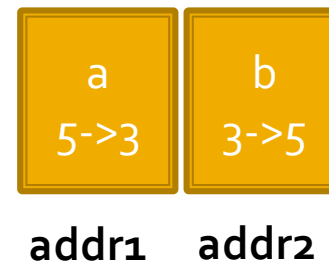
```
int main() {  
    int a = 5, b = 3;  
    swap(a, b); // 5 3  
    return 0;  
}
```



Функции

```
void swap(int* p, int* q) {  
    int temp = *p;  
    *p = *q;  
    *q = temp;  
}
```

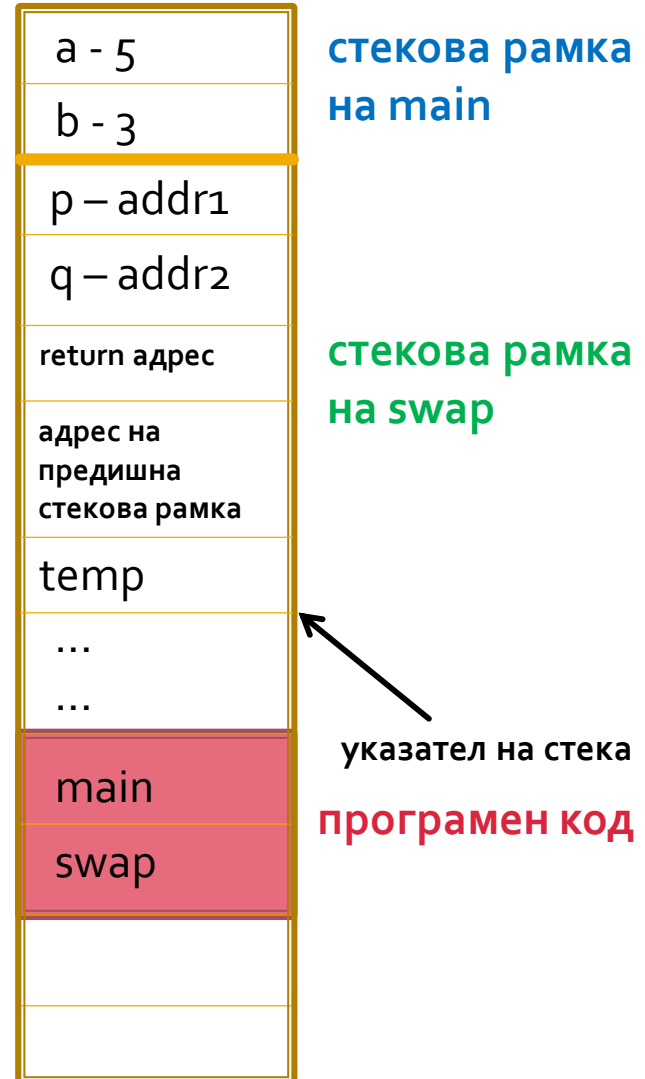
```
int main() {  
    int a = 5, b = 3;  
    swap(&a, &b); // 3 5  
    return 0;  
}
```



Функции

```
void swap(int* p, int* q) {  
    int temp = *p;  
    *p = *q;  
    *q = temp;  
}
```

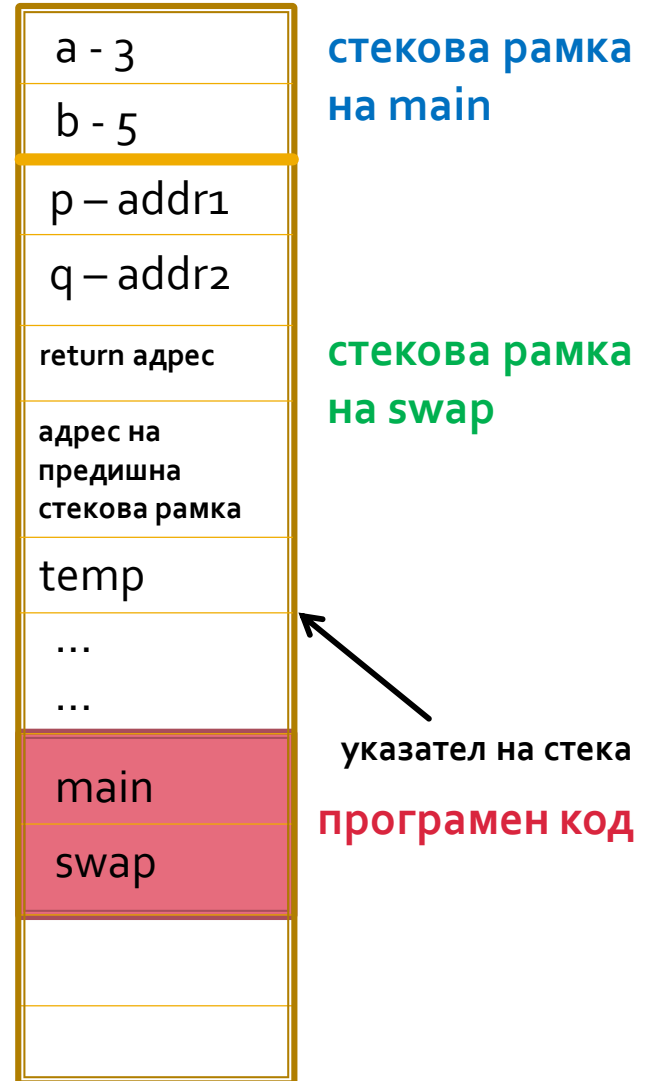
```
int main() {  
    int a = 5, b = 3;  
    swap(&a, &b); // 3 5  
    return 0;  
}
```



Функции

```
void swap(int* p, int* q) {  
    int temp = *p;  
    *p = *q;  
    *q = temp;  
}
```

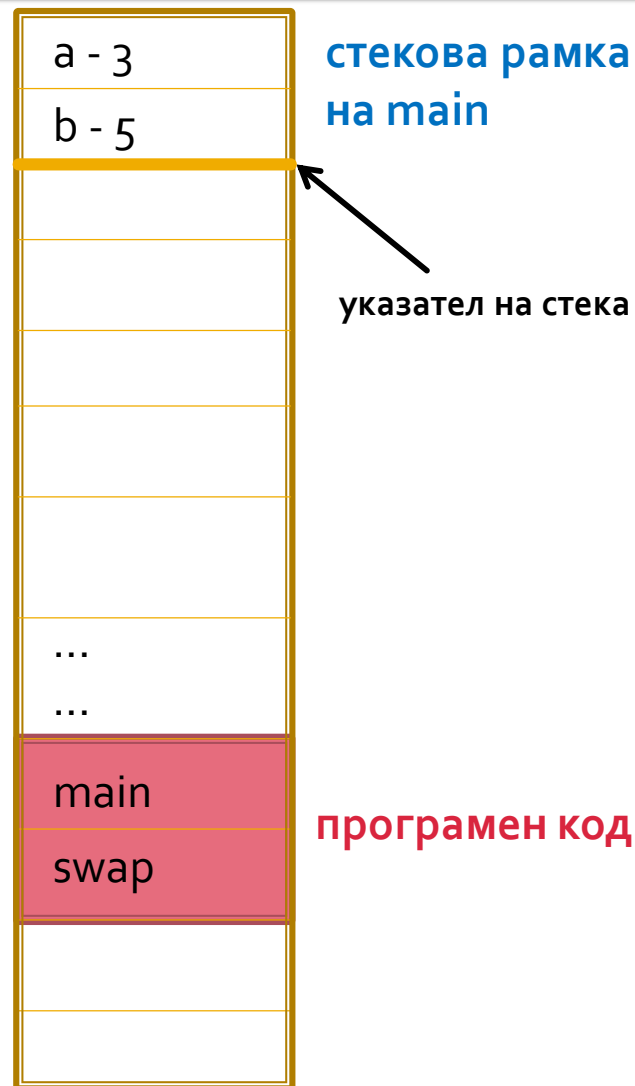
```
int main() {  
    int a = 5, b = 3;  
    swap(&a, &b); // 3 5  
    return 0;  
}
```



Функции

```
void swap(int* p, int* q) {  
    int temp = *p;  
    *p = *q;  
    *q = temp;  
}
```

```
int main() {  
    int a = 5, b = 3;  
    swap(&a, &b); // 3 5  
    return 0;  
}
```



Функции

Област на идентификатори:

- **Глобална** – дефинирани са пред всички функции, константи и променливи. Могат да се използват във всички функции, освен ако не е дефиниран локален идентификатор със същото име в някоя от функциите.
- **Локална** – дефинирани са във функция и не могат да се използват в други функции. Областта им започва от дефиницията и завършва с края на блока, в който идентификаторът е дефиниран. Локалният идентификатор „скрива“ нелокалния в областта си. Областта на формалните параметри е локална и е тялото на функцията.
- **Област на клас**

ФУНКЦИИ

```
int sum = 0;
void testFunction() {
    sum += 1;
}

void testFunction2() {
    sum += 2;
}

int main() {
    testFunction();
    testFunction2();
    cout << sum << endl;    // 3

    return 0;
}
```

ФУНКЦИИ

Пример:

```
int sum = 0;
void testFunction() {
    int sum = 10;
    sum += 1;
}

void testFunction2() {
    sum += 2;
}

int main() {
    testFunction();
    testFunction2();
    cout << sum << endl;    // 2

    return 0;
}
```

Функции

Едномерни масиви като формални параметри:

- $T \ a[]$ – формален параметър a от тип едномерен масив от тип T .
- $T^* \ a$ – формален параметър a от тип указател към тип T .

* Трябва да се предаде и размерността на масива

Функции

Да се напише функция, която въвежда елементите на масив.

```
void initArrayElements(int a[], int length) {
    for(int i = 0; i < length; i++) {
        cout << "a[" << i << "]=";
        cin >> a[i];
    }
}

int main() {
    int arr[10];
    initArrayElements(arr, 10);

    return 0;
}
```

Функции

```
void initArrayElements(int a[], int length) {  
    for(int i = 0; i < length; i++) {  
        cout << "a[" << i << "]=";  
        cin >> a[i];  
    }  
}
```

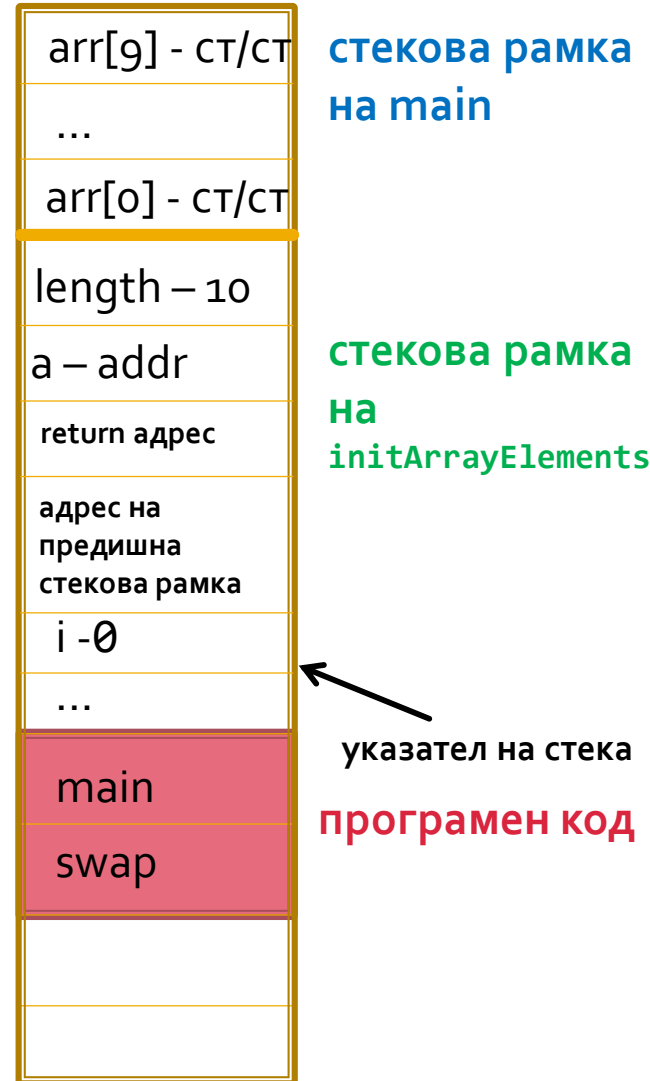
```
int main() {  
    int arr[10];  
    initArrayElements(arr, 10);  
  
    return 0;  
}
```



Функции

```
void initArrayElements(int a[], int length) {  
    for(int i = 0; i < length; i++) {  
        cout << "a[" << i << "]=";  
        cin >> a[i]; // *(a+i)  
    }  
}
```

```
int main() {  
    int arr[10];  
    initArrayElements(arr, 10);  
  
    return 0;  
}
```



Функции

```
void initArrayElements(int a[], int length) {  
    for(int i = 0; i < length; i++) {  
        cout << "a[" << i << "]=";  
        cin >> a[i]; // *(a+i)  
    }  
}
```

```
int main() {  
    int arr[10];  
    initArrayElements(arr, 10);  
  
    return 0;  
}
```



Функции

Многомерни масиви като формални параметри:

- `T matrix[][20]` – формален параметър `matrix` от тип двумерен масив от тип `T`.

В описанието му трябва да присъстват **като константи** всички размери с изключение на първия.

- `T (*matrix)[20]` – формален параметър `matrix` от тип указател към тип `T`.

** Трябва да се предаде и размерността на масива.*

Функции

Да си припомним (лекция „Масиви и указатели“):

```
// *(*matrix + i)+j)
```

Адресът на третия елемент на матрицата (int matrix[5][4])

```
*(matrix + 2) + 0;    matrix + 2;    // correct
```

```
matrix + 8 is equal to matrix + 8*sizeof(int*) // x
```

```
*matrix + 8;    matrix + 2;
```

Функции

Да се напише функция, която въвежда елементите на двумерен масив ($N \times M$). Нека $N \neq M$.

```
const int MAX_ROWS = 10;
const int MAX_COLS = 15;
void initMatrix(int matrix[][MAX_COLS], int rows, int cols) {
    for(int i = 0; i < rows; i++) {
        for(int j = 0; j < cols; j++) {
            cout << "matrix[" << i << "][" << j << "]=";
            cin >> matrix[i][j];
        }
    }
}

int main() {
    int matrix[MAX_ROWS][MAX_COLS];
    initMatrix(matrix, MAX_ROWS, MAX_COLS);

    return 0;
}
```

Функции

Да се напише функция, която въвежда елементите на двумерен масив ($N \times M$). Нека $N == M$.

```
const int MAX_ROWS_COLS = 10;
void initSquareMatrix(int squareMatrix[][MAX_ROWS_COLS], int size) {
    for(int i = 0; i < size; i++) {
        for(int j = 0; j < size; j++) {
            cout << „squareMatrix[" << i << "][" << j << "]=“;
            cin >> squareMatrix[i][j];
        }
    }
}

int main() {
    int squareMatrix[MAX_ROWS_COLS][MAX_ROWS_COLS];
    initSquareMatrix(squareMatrix, MAX_ROWS_COLS);

    return 0;
}
```

Функции

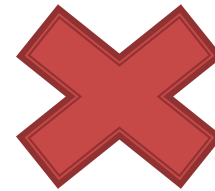
Масивите като върнати оценки:

- Функциите **не могат** да са от тип масив.
- Функции могат да бъдат от тип указател.

Функции

// Не можем да върнем масив, който се създава във функцията

```
int* initArray() {  
    int arr[MAX_LENGTH];  
    for(int i = 0; i < MAX_LENGTH; i++) {  
        cout << "arr[" << i << "]=";  
        cin >> arr[i];  
    }  
    return arr;  
}
```



* arr е в стековата рамка на initArray

ФУНКЦИИ

```
int* initArray(int arr[], int length) {  
    for(int i = 0; i < length; i++) {  
        cout << "arr[" << i << "]=";  
        cin >> arr[i];  
    }  
    return arr;  
}
```

Задача

Да се напише функция, която заменя всички срещания на символа `x` в символния низ `s` със символа `y`.

```
void replace(char *str, char x, char y) {
    int charIndex = strlen(str) - 1;

    while(charIndex >= 0) {
        if (str[charIndex] == x) {
            str[charIndex] = y;
        }
        charIndex--;
    }
}
```

Следва продължение...