

Функции

(част 3)

Трифон Трифонов

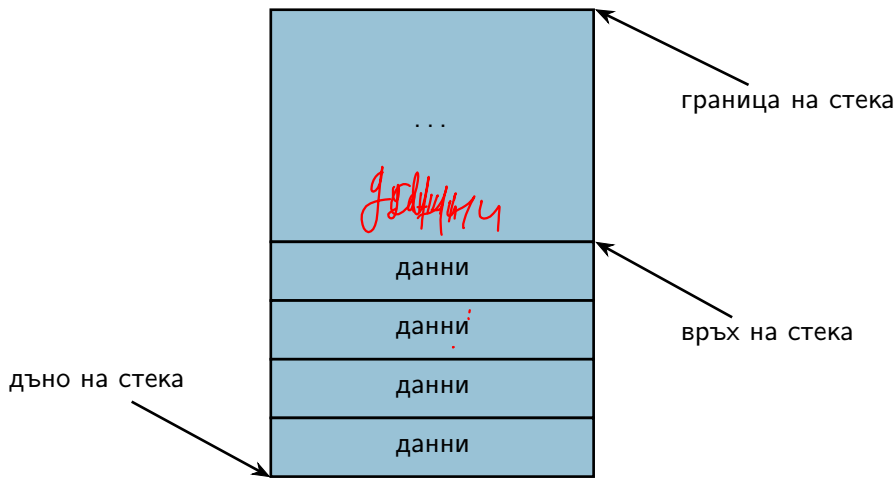
Увод в програмирането,
спец. Компютърни науки, 1 поток,
спец. Софтуерно инженерство,
2016/17 г.

14 декември 2016 г.

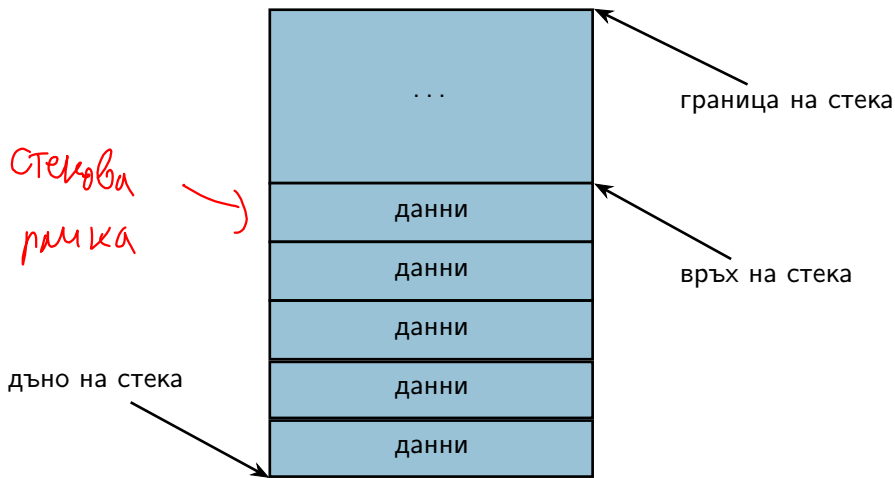
Схема на програмната памет



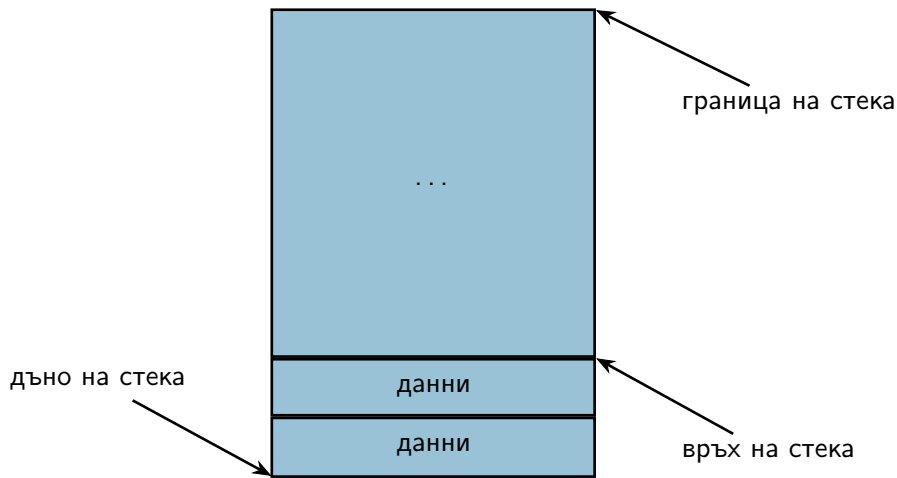
Програмен стек



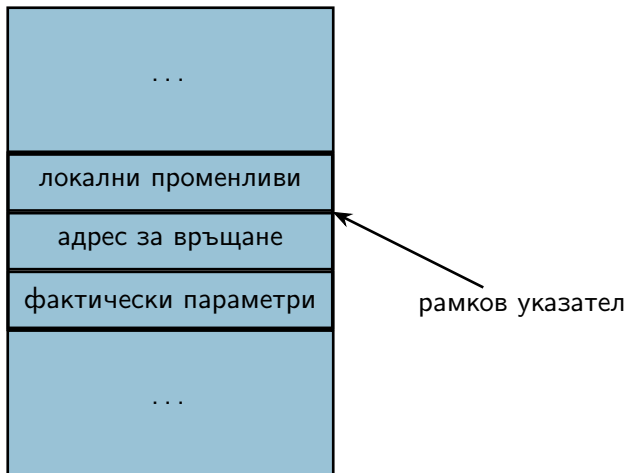
Програмен стек



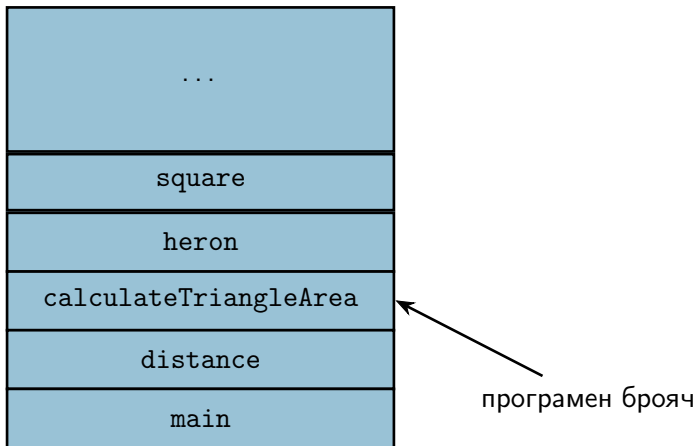
Програмен стек



Стекова рамка на функция



Област за програмен код



Предаване по стойност (call by value)

- пресмята се стойността на фактическия параметър

Предаване по стойност (call by value)

- пресмята се стойността на фактическия параметър
- в стековата рамка на функцията се създава **копие** на стойността

Предаване по стойност (call by value)

- пресмята се стойността на фактическия параметър
- в стековата рамка на функцията се създава **копие** на стойността
- всяка промяна на стойността остава локална за функцията

Предаване по стойност (call by value)

- пресмята се стойността на фактическия параметър
- в стековата рамка на функцията се създава **копие** на стойността
- всяка промяна на стойността остава локална за функцията
- при завършване на функцията, предадената стойност и всички промени над нея **изчезват**

Предаване по псевдоним (call by reference)

- Понякога искаме промените във **формалните** параметри да се отразят във **фактическите** параметри

Предаване по псевдоним (call by reference)

- Понякога искаме промените във **формалните** параметри да се отразят във **фактическите** параметри
- Тогава трябва да обявим, че искаме фактическите параметри да могат да бъдат променяни

Предаване по псевдоним (call by reference)

- Понякога искаме промените във **формалните** параметри да се отразят във **фактическите** параметри
- Тогава трябва да обявим, че искаме фактическите параметри да могат да бъдат променяни
- $\langle \text{параметър} \rangle ::= \langle \text{тип} \rangle \& \langle \text{идентификатор} \rangle$

Предаване по псевдоним (call by reference)

- Понякога искаме промените във **формалните** параметри да се отразят във **фактическите** параметри
- Тогава трябва да обявим, че искаме фактическите параметри да могат да бъдат променяни
- $\langle \text{параметър} \rangle ::= \langle \text{тип} \rangle \& \langle \text{идентификатор} \rangle$
- **Примери:**

Предаване по псевдоним (call by reference)

- Понякога искаме промените във **формалните** параметри да се отразят във **фактическите** параметри
- Тогава трябва да обявим, че искаме фактическите параметри да могат да бъдат променяни
- `<параметър> ::= <тип>& <идентификатор>`
- **Примери:**
 - `int add5(int& x) { x += 5; return x; }`

Предаване по псевдоним (call by reference)

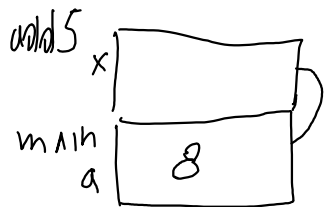
- Понякога искаме промените във **формалните** параметри да се отразят във **фактическите** параметри
- Тогава трябва да обявим, че искаме фактическите параметри да могат да бъдат променяни
- `<параметър> ::= <тип>& <идентификатор>`
- **Примери:**
 - `int add5(int& x) { x += 5; return x; }`
 - **фактическият параметър трябва да е lvalue!**

Предаване по псевдоним (call by reference)

- Понякога искаме промените във **формалните** параметри да се отразят във **фактическите** параметри
- Тогава трябва да обявим, че искаме фактическите параметри да могат да бъдат променяни
- `<параметър> ::= <тип>& <идентификатор>`
- **Примери:**
 - `int add5(int& x) { x += 5; return x; }`
 - фактическият параметър трябва да е lvalue!
 - `add5(3);`

Предаване по псевдоним (call by reference)

- Понякога искаме промените във **формалните** параметри да се отразят във **фактическите** параметри
- Тогава трябва да обявим, че искаме фактическите параметри да могат да бъдат променяни
- `<параметър> ::= <тип>& <идентификатор>`
- **Примери:**
 - `int add5(int& x) { x += 5; return x; }`
 - фактическият параметър трябва да е lvalue!
 - `add5(3);`
 - `int a = 3; cout << add5(a) << ' ' << a;`



Пример за предаване по псевдоним

Размяна на две променливи

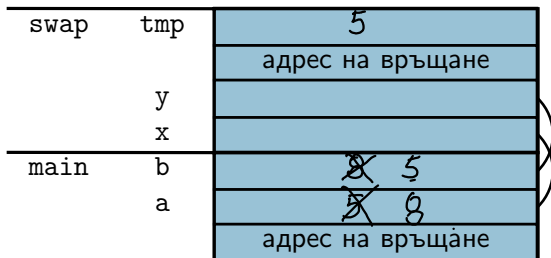
```
void swap(int& x, int& y) {  
    int tmp = x;  
    x = y;  
    y = tmp;  
}
```

Пример за предаване по псевдоним

Размяна на две променливи

```
void swap(int& x, int& y) {  
    int tmp = x;  
    x = y;  
    y = tmp;  
}  
  
int main() {  
    int a = 5, b = 8;  
    swap(a, b);  
    cout << a << ' ' << b << endl;  
}
```

Стекова рамка при предаване по псевдоним



Предаване по указател/адрес (call by pointer)

- Предава се **адрес** вместо стойност

Предаване по указател/адрес (call by pointer)

- Предава се **адрес** вместо стойност
- Фактическите параметри трябва да са от тип “указател към нещо”

Предаване по указател/адрес (call by pointer)

- Предава се **адрес** вместо стойност
- Фактическите параметри трябва да са от тип “указател към нещо”
- Функцията може да променя стойности на външни за функцията променливи **през подадените ѝ указатели**

Предаване по указател/адрес (call by pointer)

- Предава се **адрес** вместо стойност
- Фактическите параметри трябва да са от тип “указател към нещо”
- Функцията може да променя стойности на външни за функцията променливи **през подадените ѝ указатели**
- **Примери:**

Предаване по указател/адрес (call by pointer)

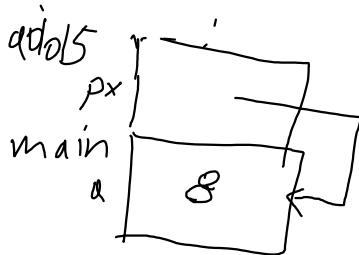
- Предава се **адрес** вместо стойност
- Фактическите параметри трябва да са от тип “указател към нещо”
- Функцията може да променя стойности на външни за функцията променливи **през подадените ѝ указатели**
- **Примери:**
 - `int add5(int* px) { *px += 5; return *px; }`

Предаване по указател/адрес (call by pointer)

- Предава се **адрес** вместо стойност
- Фактическите параметри трябва да са от тип “указател към нещо”
- Функцията може да променя стойности на външни за функцията променливи **през подадените ѝ указатели**
- **Примери:**
 - `int add5(int* px) { *px += 5; return *px; }`
 - ~~`add5(3);`~~ ~~`add5(&3);`~~

Предаване по указател/адрес (call by pointer)

- Предава се **адрес** вместо стойност
- Фактическите параметри трябва да са от тип “указател към нещо”
- Функцията може да променя стойности на външни за функцията променливи **през подадените ѝ указатели**
- **Примери:**
 - `int add5(int* px) { *px += 5; return *px; }`
 - ~~`add5(3); add5(&3);`~~
 - `int a = 3; cout << add5(&a) << ' ' << a;`



Пример за предаване по указател

Размяна на две променливи

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

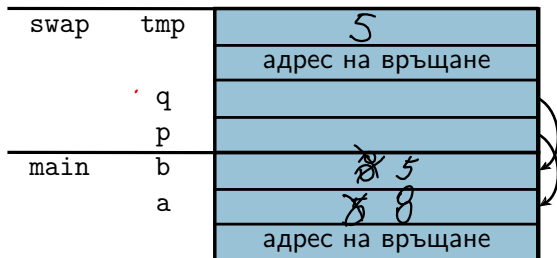

Пример за предаване по указател

Размяна на две променливи

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

int main() {
    int a = 5, b = 8;
    swap(&a, &b);
    cout << a << ' ' << b << endl;
}
```

Стекова рамка при предаване по указател



Предаване на масиви като параметри

- $\langle \text{параметър_масив} \rangle ::= \langle \text{тип} \rangle \langle \text{име} \rangle [[\langle \text{константен_израз} \rangle]]$
 $\langle \text{тип} \rangle * \langle \text{име} \rangle$

Предаване на масиви като параметри

- $\langle \text{параметър_масив} \rangle ::= \langle \text{тип} \rangle \langle \text{име} \rangle [[\langle \text{константен_израз} \rangle]]$
 $\langle \text{тип} \rangle * \langle \text{име} \rangle$
- ВСЪЩНОСТ...

Предаване на масиви като параметри

- $\langle \text{параметър_масив} \rangle ::= \langle \text{тип} \rangle \langle \text{име} \rangle [[\langle \text{константен_израз} \rangle]]$
 $\langle \text{тип} \rangle * \langle \text{име} \rangle$
- всъщност...
- ...масивите се предават **по указател!**

Предаване на масиви като параметри

- $\langle \text{параметър_масив} \rangle ::= \langle \text{тип} \rangle \langle \text{име} \rangle [[\langle \text{константен_израз} \rangle]]$
 $\langle \text{тип} \rangle * \langle \text{име} \rangle$
- всъщност...
- ...масивите се предават **по указател!**
- ...затова размерът на масива в скобите се игнорира!

Предаване на масиви като параметри

- $\langle \text{параметър_масив} \rangle ::= \langle \text{тип} \rangle \langle \text{име} \rangle \left[\left[\langle \text{константен_израз} \rangle \right] \right]$
 $\langle \text{тип} \rangle * \langle \text{име} \rangle$
- всъщност...
- ...масивите се предават **по указател!**
- ...затова размерът на масива в скобите се игнорира!
- ...затова промените в винаги се отразяват в оригинала!

Предаване на многомерни масиви като параметри

- $\langle \text{параметър_многомерен_масив} \rangle ::=$
 $\langle \text{тип} \rangle \langle \text{име} \rangle \llbracket \langle \text{константа} \rangle \rrbracket \{ \llbracket \langle \text{константа} \rangle \rrbracket \} |$
 $\langle \text{тип} \rangle (*\langle \text{име} \rangle) \{ \llbracket \langle \text{константа} \rangle \rrbracket \}$

Предаване на многомерни масиви като параметри

- $\langle \text{параметър_многомерен_масив} \rangle ::=$
 $\langle \text{тип} \rangle \langle \text{име} \rangle \llbracket \langle \text{константа} \rangle \rrbracket \{ \llbracket \langle \text{константа} \rangle \rrbracket \} |$
 $\langle \text{тип} \rangle (*\langle \text{име} \rangle) \{ \llbracket \langle \text{константа} \rangle \rrbracket \}$
- многомерните масиви също се предават по указател

Предаване на многомерни масиви като параметри

- $\langle \text{параметър_многомерен_масив} \rangle ::=$
 $\langle \text{тип} \rangle \langle \text{име} \rangle \llbracket \langle \text{константа} \rangle \rrbracket \{ \llbracket \langle \text{константа} \rangle \rrbracket \} |$
 $\langle \text{тип} \rangle (*\langle \text{име} \rangle) \{ \llbracket \langle \text{константа} \rangle \rrbracket \}$
- многомерните масиви също се предават по указател
- първата размерност се игнорира

Предаване на многомерни масиви като параметри

- `<параметър_многомерен_масив> ::=`
`<тип> <име> [[<константа>]] { [[<константа>]] } |`
`<тип> (*<име>) { [[<константа>]] }`
- многомерните масиви също се предават по указател
- първата размерност се игнорира
 - останалите трябва да се укажат, за да работи правилно указателната аритметика

Предаване на многомерни масиви като параметри

- $\langle \text{параметър_многомерен_масив} \rangle ::=$
 $\langle \text{тип} \rangle \langle \text{име} \rangle \llbracket \langle \text{константа} \rangle \rrbracket \{ \llbracket \langle \text{константа} \rangle \rrbracket \} |$
 $\langle \text{тип} \rangle (*\langle \text{име} \rangle) \{ \llbracket \langle \text{константа} \rangle \rrbracket \}$
- многомерните масиви също се предават по указател
- първата размерност се игнорира
 - останалите трябва да се укажат, за да работи правилно указателната аритметика
- (поне) първата размерност трябва да се подава като параметър

Предаване на многомерни масиви като параметри

- `<параметър_многомерен_масив> ::=`
`<тип> <име> [[<константа>]] { [[<константа>]] } |`
`<тип> (*<име>) { [<константа>] }`
- многомерните масиви също се предават по указател
- първата размерност се игнорира
 - останалите трябва да се укажат, за да работи правилно указателната аритметика
- (поне) първата размерност трябва да се подава като параметър
- **Внимание:** `int* a[10]` е различно от `int (*a)[10]`!

Предаване на многомерни масиви като параметри

- `<параметър_многомерен_масив> ::=`
`<тип> <име> [[<константа>]] { [<константа>] } |`
`<тип> (*<име>) { [<константа>] }`
- многомерните масиви също се предават по указател
- първата размерност се игнорира
 - останалите трябва да се укажат, за да работи правилно указателната аритметика
- (поне) първата размерност трябва да се подава като параметър
- **Внимание:** `int* a[10]` е различно от `int (*a)[10]`!
 - `int* a[10]` \iff масив от 10 указателя към цели числа

Предаване на многомерни масиви като параметри

- $\langle \text{параметър_многомерен_масив} \rangle ::=$
 $\langle \text{тип} \rangle \langle \text{име} \rangle \llbracket \langle \text{константа} \rangle \rrbracket \{ \llbracket \langle \text{константа} \rangle \rrbracket \} |$
 $\langle \text{тип} \rangle (*\langle \text{име} \rangle) \{ \llbracket \langle \text{константа} \rangle \rrbracket \}$
- многомерните масиви също се предават по указател
- първата размерност се игнорира
 - останалите трябва да се укажат, за да работи правилно указателната аритметика
- (поне) първата размерност трябва да се подава като параметър
- **Внимание:** `int* a[10]` е различно от `int (*a)[10]`!
 - `int* a[10]` \iff масив от 10 указателя към цели числа
 - `int (*a)[10]` \iff указател към масив от десет цели числа

Предаване на многомерни масиви като параметри

- $\langle \text{параметър_многомерен_масив} \rangle ::=$
 $\langle \text{тип} \rangle \langle \text{име} \rangle \llbracket \langle \text{константа} \rangle \rrbracket \{ \llbracket \langle \text{константа} \rangle \rrbracket \} |$
 $\langle \text{тип} \rangle (*\langle \text{име} \rangle) \{ \llbracket \langle \text{константа} \rangle \rrbracket \}$
- многомерните масиви също се предават по указател
- първата размерност се игнорира
 - останалите трябва да се укажат, за да работи правилно указателната аритметика
- (поне) първата размерност трябва да се подава като параметър
- **Внимание:** `int* a[10]` е различно от `int (*a)[10]`!
 - `int* a[10]` \iff масив от 10 указателя към цели числа
 - `int (*a)[10]` \iff указател към масив от десет цели числа
 - ...но понеже масивите от тип T могат да се разглеждат като указатели към тип T...

Предаване на многомерни масиви като параметри

- $\langle \text{параметър_многомерен_масив} \rangle ::=$
 $\langle \text{тип} \rangle \langle \text{име} \rangle \llbracket \langle \text{константа} \rangle \rrbracket \{ \llbracket \langle \text{константа} \rangle \rrbracket \} |$
 $\langle \text{тип} \rangle (*\langle \text{име} \rangle) \{ \llbracket \langle \text{константа} \rangle \rrbracket \}$
- многомерните масиви също се предават по указател
- първата размерност се игнорира
 - останалите трябва да се укажат, за да работи правилно указателната аритметика
- (поне) първата размерност трябва да се подава като параметър
- **Внимание:** `int* a[10]` е различно от `int (*a)[10]`!
 - `int* a[10]` \iff масив от 10 указателя към цели числа
 - `int (*a)[10]` \iff указател към масив от десет цели числа
 - ...но понеже масивите от тип T могат да се разглеждат като указатели към тип T...
 - `int (*a)[10]` \iff масив от масив от десет цели числа

Предаване на многомерни масиви като параметри

- $\langle \text{параметър_многомерен_масив} \rangle ::=$
 $\langle \text{тип} \rangle \langle \text{име} \rangle \llbracket \langle \text{константа} \rangle \rrbracket \{ \llbracket \langle \text{константа} \rangle \rrbracket \} |$
 $\langle \text{тип} \rangle (*\langle \text{име} \rangle) \{ \llbracket \langle \text{константа} \rangle \rrbracket \}$
- многомерните масиви също се предават по указател
- първата размерност се игнорира
 - останалите трябва да се укажат, за да работи правилно указателната аритметика
- (поне) първата размерност трябва да се подава като параметър
- **Внимание:** `int* a[10]` е различно от `int (*a)[10]`!
 - `int* a[10]` \iff масив от 10 указателя към цели числа
 - `int (*a)[10]` \iff указател към масив от десет цели числа
 - ...но понеже масивите от тип T могат да се разглеждат като указатели към тип T...
 - `int (*a)[10]` \iff масив от масив от десет цели числа
 - `int (*a)[10]` \iff двумерен масив от цели числа с 10 колони

Примерни функции

- 1 Да се напише функция, която извежда матрица от числа

Примерни функции

- 1 Да се напише функция, която извежда матрица от числа
- 2 Да се напише функция, която въвежда масив от низове

3 n
abc \n
def \n

Примерни функции

- 1 Да се напише функция, която извежда матрица от числа
- 2 Да се напише функция, която въвежда масив от низове
- 3 Да се напише функция, която проверява дали дадена дума се съдържа в масив от низове

Примерни функции



- 1 Да се напише функция, която извежда матрица от числа
- 2 Да се напише функция, която въвежда масив от низове
- 3 Да се напише функция, която проверява дали дадена дума се съдържа в масив от низове
- 4 Да се напише функция, която умножава две правоъгълни матрици

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}$$

Указателите като върнат резултат

Основно правило: трябва да осигурим, че винаги връщаме указатели към обекти, които ще продължат да съществуват след като функцията приключи работа.

Указателите като върнат резултат

Основно правило: трябва да осигурим, че винаги връщаме указатели към обекти, които ще продължат да съществуват след като функцията приключи работа.

Пример:

```
int* pointMax(int* p, int* q) {  
    if (*p > *q)  
        return p;  
    return q;  
}  
  
...  
int* r = pointMax(&a, &b); (*r)--;
```


Псевдонимите като върнат резултат

Ваши същото правило като за указателите: връщаме псевдоними към обекти, които ще останат “живи”.

Псевдонимите като върнат резултат

Ваши същото правило като за указателите: връщаме псевдоними към обекти, които ще останат “живи”.

Пример:

```
int& middle(int& x, int& y, int& z) {  
    if (x <= y && y <= z || z <= y && y <= x)  
        return y;  
    if (y <= z && z <= x || x <= z && z <= y)  
        return z;  
    return x;  
}  
...  
middle(a, b, c) = 5;
```

Масивите като върнат резултат

- Функциите **не могат** да имат “масив от T” като тип на резултата

Масивите като върнат резултат

- Функции **не могат** да имат “масив от T” като тип на резултата
- ...но могат да имат тип на резултата “указател към T”

Масивите като върнат резултат

- Функции **не могат** да имат “масив от T” като тип на резултата
- ...но могат да имат тип на резултата “указател към T”
- по този начин функциите могат да връщат като резултат **едномерни масиви**

Масивите като върнат резултат

- Функции **не могат** да имат “масив от T” като тип на резултата
- ...но могат да имат тип на резултата “указател към T”
- по този начин функциите могат да връщат като резултат **едномерни масиви**
- **Внимание:** връщат се само масиви, които ще продължат да съществуват след като функцията завърши

Масивите като върнат резултат

- Функции **не могат** да имат “масив от T” като тип на резултата
- ...но могат да имат тип на резултата “указател към T”
- по този начин функциите могат да връщат като резултат **едномерни масиви**
- **Внимание:** връщат се само масиви, които ще продължат да съществуват след като функцията завърши
- **Примери:**

Масивите като върнат резултат

- Функции **не могат** да имат “масив от T” като тип на резултата
- ...но могат да имат тип на резултата “указател към T”
- по този начин функциите могат да връщат като резултат **едномерни масиви**
- **Внимание:** връщат се само масиви, които ще продължат да съществуват след като функцията завърши
- **Примери:**
 - Да се реализира `strchr`

Масивите като върнат резултат

- Функциите **не могат** да имат “масив от T” като тип на резултата
- ...но могат да имат тип на резултата “указател към T”
- по този начин функциите могат да връщат като резултат **едномерни масиви**
- **Внимание:** връщат се само масиви, които ще продължат да съществуват след като функцията завърши
- **Примери:**
 - Да се реализира `strchr`
 - Да се реализира `strstr`

Масивите като върнат резултат

- Функциите **не могат** да имат “масив от T” като тип на резултата
- ...но могат да имат тип на резултата “указател към T”
- по този начин функциите могат да връщат като резултат **едномерни масиви**
- **Внимание:** връщат се само масиви, които ще продължат да съществуват след като функцията завърши
- **Примери:**
 - Да се реализира `strchr`
 - Да се реализира `strstr`
 - Да се реализира функция, която връща позицията на първото различие между два низа