



## Уп - преговор

### Тема 1: Типове данни

#### 1. Числови

a) int  $\rightarrow -2^{31}, \dots, 2^{31}-1 \rightarrow 4 \text{ bytes}$

short  $\rightarrow -2^{15}, \dots, 2^{15}-1 \rightarrow 2 \text{ bytes}$

long long  $\rightarrow -2^{63}, \dots, 2^{63}-1 \rightarrow 8 \text{ bytes}$

unsigned int  $\rightarrow 0, \dots, 2^{64}-1$

#### 2. С плаваща запетая

- float  $\rightarrow 4 \text{ bytes}$  } не са избрани, затова

- double  $\rightarrow 8 \text{ bytes}$  } използваме в краен  
случай.

#### 3. Символи

- char  $\rightarrow 1 \text{ byte} \rightarrow -128, \dots, 127$

\* използват се ' ', за да не се обръща  
стойността на променливата с друга  
променлива, зададена в кода.

- ASCII table,

#### 4. Преобразуване на типове данни

a) без загуба на информация

$\rightarrow \text{int} \rightarrow \text{float}$

$\rightarrow \text{char} \rightarrow \text{int}, \text{float}$

$\rightarrow \text{bool} \rightarrow \text{char}, \text{int}, \text{float}$

- 5) сос жагыбы на инфоормация
- $\rightarrow \text{float} \rightarrow \text{int, bool, char}$
  - $\rightarrow \text{int} \rightarrow \text{char, bool}$
  - $\rightarrow \text{char} \rightarrow \text{bool}$ .

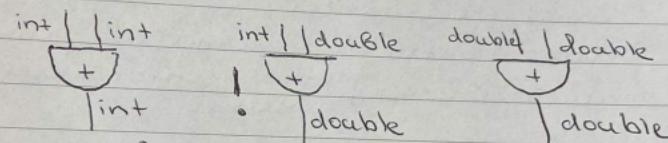
### 5. Оператори

#### a) арифметички

$\rightarrow "+" \rightarrow$  ишане целочислен ч көрсеткіштейн

$\rightarrow "-" \rightarrow$

$\rightarrow "*" \rightarrow$



$\rightarrow "/" \}$  давам пълноценно деление при

$\rightarrow "%" \}$  целочислен ч данни.

**⚠️** Арифметичките оператори връщат стойност!

#### б) за сравнение

$\rightarrow ">", ">=", "<", "<=", "=" , "!="$

**⚠️** Не сравняване float ч double с max директно, защото иначе нечестности в паметта!

$$\text{abs}(a-b) \leq 10e^{-6} \Leftrightarrow a = b$$

$$a - b \leq 10e^{-6} \Leftrightarrow a \leq b$$

$$a - b \geq 10e^{-6} \Leftrightarrow a \geq b$$



⚠ Операторите за сравнение връщат  
Bool стойност!

в) логически оператори

$\rightarrow "!" \rightarrow$  отрицание

$\rightarrow "||" \rightarrow$  логическо "и"

$\rightarrow "||" \rightarrow$  логическо "или"

⚠ Логическите оператори връщат  
булеви стойности!

г) приоритет на оператори

постфиксни (++ , --) > префиксни (++ , --) >  
 $(* , //) > (+ - \% ) > (> <) > (\&\&) > (||) >$   
тернарен > (=) > (, )

д) оператор за присвояване " $=$ "

$\rightarrow$  двоносочувствен

$\rightarrow$  връща променлива заедно с

$"+=", "+-", "-=", ", ", "/=", "*=".$

е) Постфиксен и префиксен ++:

$\rightarrow a++ \rightarrow$  връща стойност (стара) и

след това се увеличава.  $\rightarrow rvalue$

$\rightarrow ++a \rightarrow$  връща променлива и се увели-  
чава едновременно с това.  $\rightarrow lvalue$

V



$a++$   $\rightarrow$  не се кончира  
връчка стойност



int a = 5;

$a += (a++) + (a++) ;$

$cout << a;$   $\rightarrow$  принтира 14, защото  
а се занесства със + и възвието  
обединени



int a = 6;

$a = (a = 3) + (a = 4) ;$

$cout << a;$   $\rightarrow$  Връчка 8.

## Тема 2: Условни конструкции

1. if (<буково условие>) {  
    body  
}

else if (<буково условие>){  
    body  
}

⋮

else {  
    body  
}

2. switch (`bool / char / int`) {  
    case `x`: ... `break`;  
    :  
    default: ... `break`;

3

3. Тернарен оператор

⚠ Винаги връчва стойност!

<Булево усв>? <израз1> : <израз2>;

Δ Израз1 и Израз2 трябва

да връщат един и същи тип  
или да могат да се преобразу-  
ват директно!

Тема 3: Числа

1. while (<булево условие>) {  
    <body>

3

⚠ Алгоритъм на Евклид (за НОД):

`int a, b;`

`cin >> a >> b;`

`while (b != 0) {`

`int c = a % b;`

`a = b;`

`b = c;`

`cout << a;`



## 2. For - цикъл

```
for (<инициализация>; <бул. усл>; <актива-  
ция>) {  
    <тело>  
}
```

- ⚠ Извършва се в последователност  
 $A \rightarrow B \rightarrow D \rightarrow C \rightarrow B \rightarrow D \rightarrow C \dots \rightarrow B$

## 3. Нивен цикъл на променлива

⚠ Една променлива "ниве" от своето деклариране до края на най-вътрешния scope, която сме дефинирали.

⚠ Винаги следи да декларираме променливите ВЪВНОННО и АЙ-КОСНО.

## 4. Оператори break и continue

- a) break  $\rightarrow$  спира изпълнението на цикъла  
b) continue  $\rightarrow$  спира текущата итерация и връща обратно на проверката в началото на цикъла.

## 5. Водеща променлива за цикъл:

⚠ Това е променлива, която на читателя си смека ст-ста и се използва само в рамките на цикъла.



=> for = while + водеща променлива.

## Тема 4: Функции

### 1. Def

а) математическа → систематичен начин на реализация

б) в програмирането → код, който може да е произволно използване

### 2. Синтаксис

<тип на връщане> <име> (<параметри>)  
{ <тело> }

### 3. Оператор return:

⚠ Спира изпълняването на функцията и замества извикването ѝ с подадения резултат.

4. ⚡ Функциите приемат копия на параметрите си!

### 5. Правила!

- проверка за входа;
- generic функции;

В) вложени цикли - НЕ!,  $\rightarrow$  изнасяне в отделна функция.

Г) int main()  $\rightarrow$  стартираща точка на програмата. В края запицваме return 0, защото 0 проверява дали програмата е изпълнена успешно.

 Ъ-чупите ни дават по-добра абстракция, т.е. използване готови интерфейси без да трябва да дефинирате ги отново.

## Тема 5: Редирекция (препратка)

1. Def: друго име на все съществуваща променлива.

 приема l-value

int a = 5; }  
int &B = a; } 5  
a, B

B++;

cout << a;  $\rightarrow$  изкарва 6

 int & ref няма да се компилира, защото му трябва стойност.



⚠ Подаване по редференция във функции  $\rightarrow$  не се прави копие на променливата, а се подава самата променлива и изменение на върхните на функцията!

⚠ Връщане 1 нещо  $\rightarrow$  или само по редференция, или чрез ф-я.

⚠ Връщане  $\geq 2$  неща  $\rightarrow$  void  $\rightarrow$  няма тип на връщане, или чрез редференции.

## 2. Стекова рамка

а) Системен стек - матрица, по която са организирани ф-чните на нашата програма

б) Стекова рамка  $\rightarrow$  има достъп до най-горния елемент  
 $\rightarrow$  FILO принцип  $\rightarrow$  първия добавен е последният изхнат.

## 3. Комуникация между ф-чните:

$\rightarrow$  чрез връщане на резултат;

$\rightarrow$  чрез подаване на параметър по редференция;

$\rightarrow$  чрез глобални променливи  $\rightarrow$  състои в отделно място извън стека.

4. int f (...) → времща стойност  
int& f(int&a) } времща променливата  
; }  
return a-1;

Пример: симулране на `++` и `++`  
чрез ф-ция.

5. Презаписване на функции / function  
overloading → няколко функции с едно  
и същите.

⚠ Можем да презаписваме само когато  
в различните ф-ции подаваме различни  
параметри!

⚠ Типичен пример за полиморфизъм →  
зад едно име са съответни функционал-  
ности. ("`+`" и "`-`" са полиморфни).

## Тема 6: Масиви (стартични)

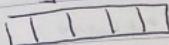
1. Def: Последователност от променливи  
от един и същи тип. Има предварително  
известна дължина (преди компилиация).



## 2. Синтаксис

`<min> <max>`

`arr`



[размер]

↓  
препар const инициализация (число).

value (число).

a) достъп до елементите

$\rightarrow \text{arr}[i] \rightarrow$  достъп до  $i$ -ия елемент на масива

⚠  $i$  започва от 0, защото показва броя на от欠缺ванията от началото на масива.

⚠ `cout << arr`  $\rightarrow$  отпечатва адрес на масива

⚠ `cout << arr[i]`  $\rightarrow$  отпечатва харбуша в дадената клетка.

⚠ В масивите има константен достъп до елементи.

б) намиране на адрес на  $i$ -та клетка:  
адрес на  $\text{arr}[i] = \text{arr} + i * \text{sizeof(int)}$

в) подаване на масив във функция.  
`void f (int arr[], size_t size);`

⚠ Не правен подаване аргумент по редференция, защото във `f` се прави копие на структурата.



int arr[3];

cout << arr[99] → ще изкара каша  
int arr[3]; } не се  
arr[99] = 4; } компи-  
лира като описани  
в паметта.

⚠️ Ако не променяме масива, то приенане  
като const. ще се дължи.

⚠️ Решето на Ератостен → важен  
алгоритъм за намаляване на търсих  
числа в даден интервал.

## Тема 7: Броени системи и побитови операции

1. Броена система за брояне

- десетична (decimal)

- двоична (Binary)

- hexadecimal → шестнадесетична

- октадична → наст-наска (само от 1).

⚠️ Ом k-нска в десетична →

$$121(3) \rightarrow ?(10)$$

$$1 \cdot 3^0 + 2 \cdot 3^1 + 1 \cdot 3^2 = 1 + 6 + 9 = 16$$

$$\text{rest} = (121 \% 10)^* \text{ mult}$$

"  
k

$$k^* = k;$$



▲ От измислена б. к-чина:

$$137_{(10)} \rightarrow ?_{(4)}$$

$$134 \% 4 = 1 \rightarrow \text{res} + = (137 \% 4)^* \text{mult}$$

$$137 / = k;$$

$$\text{mult}^* = 10;$$

▲  $10_{(n)} = k_{(10)}$

$$n(x) = P(10)$$

$$n.0_{(n)} = P.k_{(10)}$$

$$n = \boxed{\dots\dots} = \left\lfloor \frac{P}{k} \right\rfloor, \text{ закръглено надолу}$$

2. Представяне на отрицателни числа в двоична бройка с-ма-

-идая 1: първият бит да показва дали числото е положително (0) или отрицателно (1)

Проблем: 2 нули  $\rightarrow$  1000  
0000

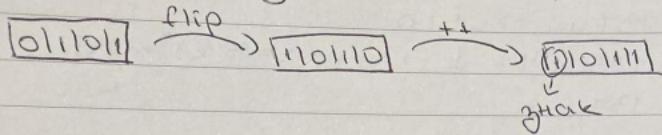
-идая 2: one's complement  $\rightarrow$  с flip-ване

5  $\boxed{110101}$

-5  $\boxed{110110}$

Проблем: 2 нули, аритметиката не използва

-нега 3 (Валидна): two's complement



⚠ Знаков түнштіктер көмкеме:

$$[-2^{k-1}; 2^{k-1} - 1]$$

Без знак:

$$[0; 2^k - 1]$$

3. Побитовы операции

a) & побитово "и"

$$1 \& 0 \rightarrow 0$$

$$0 \& 1 \rightarrow 0$$

$$0 \& 0 \rightarrow 0$$

$$1 \& 1 \rightarrow 1$$

b) | -> побитово "или"

$$0 | 0 \rightarrow 0$$

$$0 | 1 \rightarrow 1$$

$$1 | 0 \rightarrow 1$$

$$1 | 1 \rightarrow 1$$

b)  $\wedge \rightarrow \times 0\Gamma \rightarrow$  побитово извл. или

The diagram illustrates the AND operation using binary multiplication. It shows four binary digits: 0, 1, 0, 1. A bracket groups the first and third digits (0 and 1), and another bracket groups the second and fourth digits (1 and 0). An arrow labeled " $\wedge$ " points from the first group to the result 0. A bracket labeled " $\times 0\Gamma$ " points from the second group to the result 0. Below the digits, a bracket labeled "0" points to the final result 0.



1) отчестване (bitwise shift)

$\ll$  -> отчестване наляво

$\gg$  -> отчестване надясно

a = 11100110110

$a \gg 2 \rightarrow$  10101110010

избягат нови нули

⚠ От дясната страна на  $\gg$  и  $\ll$  винаги стои положително число.

⚠ Имате оператори  $\&=$ ,  $|=$ ,  $^=$ ,  $>>=$ ,  $\ll=$ , които променят левия параметър. (връщат l-value).

⚠  $2^n \rightarrow 1 \ll n;$

⚠ i-тият бит в число:

return  $n \& (1 \ll i) == (1 \ll i);$

⚠ i-тият бит да е стапе 0:

използване  $\sim$ ->invert  $\sim 101 = 010$

mask  $\sim (1 \ll i) \rightarrow$  return  $n \& mask.$

⚠ i-тият бит да е стапе 1

return  $n | (1 \ll i);$

⚠ число е четно  $\rightarrow$  return  $(n \& 1);$



Разлика между && и & (null):

&& спира, ако срещне 0

& съмта +.

### Тема 8: Указатели и string-ове

1. Указател - променлива, която държи адресът към друга променлива.

int \*ptr

тук, не е оператор

⚠ Оператор \* -> приема указател и връща променливата (десериализира).  
 $\text{cout} \ll *ptr \rightarrow$  отпечатва стойност  
 $\text{cout} \ll ptr \rightarrow$  отпечатва адрес

⚠ Разлика между указател и редференция:

- редференцията задължително се инициализира:

$\rightarrow \text{int } \&a;$ ; -> не е валидно;  $\text{int } *a;$  -> валидно.

• указателите имат неизменна стойност  
 $\rightarrow \text{nullptr} \rightarrow$  същ като нищо.

- указателите могат да менят променливите, като същ като:

$\text{int } *ptr = \&b;$ ; -> същ като b

$ptr = \&a;$ ; -> все същ като a

⚠ Приемане масиви като указатели като първия елемент.

`const int* arr;`

`arr++` → изменява указателя;  
увеличава адреса със sizeof(  
bytes).

`*arr += 5` → увеличава стойността на клетката

`(*arr + 5)` → увелич. указателя (отнесъвай)

⚠ `const int* ptr` ни дава възможност  
да променяме само ~~данные~~ данните.

`int* const ptr` → не променяме ptr,  
само данните.

2. Стингове → масив от тип `char`

→ `char* arr;`

`cout << arr;` → отпечатва елементите

⚠ Ако накрая не сме задали термина-  
рния нула '\0', то ще има проблеми за  
отпечатвана слугатки символи.

⇒ За стинг с дължина n ни трябва  
масив с дължина n+1.

⚠ Стинговете са единствените  
масиви, които не подавате във функции  
без дължина (коинцидент намерих).

 Приемане на масив като указатели като първия елемент.

`const int* arr;`

`arr++` → изменява указателя;  
увеличава адреса със sizeof(  
bytes).

`*arr += 5` → увеличава стойността на хлятката

`(*arr + 5)` → увелич. указателя (отнесъвай)

 `const int* ptr` ни дава възможност  
да променяме само ~~данные~~ <sup>помимо</sup>.

`int* const ptr` → не променяме ptr,  
само данните.

2. Стингове → масив от тип `char`

→ `char* arr;`

`cout << arr;` → отпечатва елементите

 Ако накрая не сме задали термина-  
рица nulla '\0', то ще има проблеми за  
отпечатвана слугатки символи.

⇒ За стинг с дължина n ни трябва  
масив с дължина n+1.

 Стинговете са единствените  
масиви, които не подавате във функции  
без дължина (коинцидент намерих).

⚠ Може да пропуснете 1<sup>във</sup> аргумент при посочване на размерността на матрицата.

## Тема 10: Динамична памет.

### 1. Основни типове памет

→ stack → статична

→ heap → динамична

2. Stack → паметта за стека се заделя по време на компиляцията и нейния размер е предварително известен.

⚠ При стека паметта се трие автоматично след приключване на програмата.

3. Heap → външен ресурс /памет/, която може да се заделя след компиляция, т.е. по време на изпълнение на програмата.

4. Заделане на динамична памет  
int \*ptr = new int[n]; → и може да не е пред. известна.

зададен в стека указ. към нач-кот масива.

броя на указател към началото на заделената памет.



## 5. Триене / освобождаване на памет

→ тие го извършване чрез оператор  
delete [] ;

delete [] p++ ; → освобождава дин.  
памет, която която съди p++ .

⚠ Самият указател не се трие.

⚠ За тази предизвиква памет leak / останка  
delete . В противен случай се  
предизвиква memory leak / останка  
на паметта, т.е. тя си остава  
резервирана и в един момент може  
да събрши .

⚠ В други езици има garbage  
collector , който често автоматично  
чири . памет .

⚠ Работата с динамична памет е  
по-бавна, затова за предпоследните с  
работата със статичната .

## Тема 11: Работа с динамични масиви от динамични масиви

### 1. Създаване на матрица

char\*buff[1024];

```
cin.getline(buff, 1024);
```

players Names Are [ : ] = new char [strlen(buff)+1]

```
strcpy(pl->[i],buff);
```

## Тема 12: Анализ на алгоритми . Алгоритми за търсене и сортиране.

→ 2 вида алгоритмов:

- За търсене
  - за сортиране

## 1. Сложност на алгоритма

→ no време → no-матично, замусома насті  
се купува най-швидко  
безмислено

- брати членки → best-case, average-  
case, worst-case ⇒ єп-як юзажимка  
на експериментах.

$\downarrow$   
най-лекко  
є смета



### Алгоритми за търсене:

1) Linear search

- best-case - 3 съмнени

- worst-case -  $3n+1$

**⚠** Не е важно колко стъпки премине алгоритът, а как нараства ф-ята при нарастване на входа.

**⚠**  $\Theta(f) \rightarrow$  нин-всомо от  $f$ , която

1) може да е бърза, колкото  $f$ .

$$3n+1 \asymp n$$

$$\lim_{n \rightarrow \infty} \frac{f}{g} = \begin{cases} \infty \rightarrow f \succ g \\ \text{const} (\neq 0) \rightarrow f \asymp g \\ 0 \rightarrow f \prec g \end{cases}$$

$\Theta(n) \rightarrow$  average case.

2) Binary search  $\rightarrow$  търсене на елемент

в сортиран масив

**⚠**  $mid = left + (right - left) / 2;$ ,

за да не надскочи между данни

3) проверки:

$= mid \Rightarrow$  напечати със

$> mid \Rightarrow left = mid + 1;$

$< mid \Rightarrow right = mid - 1;$

} OKATO

}  $left \leq right$



$$\begin{array}{ccc} \text{B-C} & \text{A-C} & \text{W-C} \\ \Theta(1) & \Theta(\log(n)) & \Theta(\log(n)). \end{array}$$

### **⚠ Linear vs Binary**

$$\lim_{n \rightarrow \infty} \frac{n}{\log n} = \lim_{n \rightarrow \infty} n = +\infty \Rightarrow \log(n) \Rightarrow \text{Binary}$$

Алгоритми за мисълно сортиране

1) **Bubble Sort**  $\rightarrow$  разходка по масив

! Съег k-мама разходка нак. k елем.  
са на несмама си.

```
for (int i=0; i<size; i++) { ← bool isSwapped = false;
    for (int j=0; j<size-1-i; j++) {
        if (arr[j] > arr[j+1])
            swap (arr[j], arr[j+1]);
        isSwapped = true;
    }
    if (!isSwapped)
        return;
}
```

}



$$\begin{array}{ccc} \text{B-C} & \text{A-C} & \text{W-C} \\ \Theta(n) & \Theta(n^2) & \Theta(n^2) \end{array}$$



⚠ Стабилност → запазване на  
относителната наредба → важно  
при сортиране на новите ключове!  
⇒ Bubble Sort е стабилен!

2) Selection Sort → прави минимален  
брой swap-ове, но не е  
стабилен!

```
int getMinElIndex(const int* arr,  
size_t size) {
```

```
    int minInd = 0;  
    for (int i = 1; i < size; i++) {  
        if (arr[i] < arr[minInd])
```

```
            minInd = i;
```

```
    return minInd;
```

```
}
```

```
void SelectionSort(int* arr, size_t  
size) {
```

```
    for (int i = 0; i < size - 1; i++) {
```

```
        int minInd = i + getMinElIndex  
(arr[i+1], size);
```

```
        if (minElIndex != i)
```

```
            swap(arr[minElIndex],  
                  arr[i]);
```

```
}
```



△ B-C

$\Theta(n^2)$

A-C

$\Theta(n^2)$

W-C

$\Theta(n^2)$

3) Insertion Sort → сортирует  
неч. элемент и то "insert"-виде  
на наименьшее из; сдвигает!

```
void insert (int* arr, size_t size) {
    int el = arr [size - 1];
    int iter = size - 2;
    while (iter >= 0 && el < arr [iter]) {
        arr [iter + 1] = arr [iter];
        iter--;
    }
    arr [iter + 1] = el;
}
```

}

void Insertion Sort (int\* arr, size\_t  
size) {

```
for (int i = 1; i < size; i++) {
    insert (arr, i + 1)
}
```

}



△ B-C

$\Theta(n)$

A-C

$\Theta(n^2)$

W-C

$\Theta(n^2)$

→ устанавливается, какого же эл.-са  
будет гр нестабильной эн.



## Тема 13: Функции от низок ред

1. def: ф-я, която приема (или връща)  
друга функция.

→ Синтаксис:  
`f ( ..., друга ф-я) или f(...){return;}`

2. Кандра ф-я:  
`f ( ..., [ ] (char ch, int a) { return  
a; } );`

анонимна ф-я - използва  
се, за да се подаде параметър.

тук уточняване кои  
параметри са от scope-a на  
анонимната ф-я да се  
"връщат".

Ако подадем &, то  
връщат се същите  
scope.



## Тема 14: Рекурсия

1. def: Ф-я, която извиква се си, но с други параметри

⚠ Тозин каква е рекурсивната декомпозиция на проблема и как се решава чрез по-малки инстанции на същата задача!

⚠ Да следим за нови членове на рекурсията!

### Задачи:

1) n!

unsigned fact (unsigned n)

{

if ( $n == 0$ )

return 1;

return  $n * \text{fact}(n-1)$ ;

}

2) фибоначи

int fibb (unsigned n) {

if ( $n == 0 || n == 1$ )

return 1;

return fibb(n-1) + fibb(n-2);



**Проблем:** сжатие одно и то же  
несколько раз!

**Решение:** Воведение кеша с везде  
сменяющимися стойностями  
=> memoisation

3) степенуване  $\rightarrow \Theta(n)$

if ( $x == 0$ )

return 1;

return  $n * \text{power}(n, x-1)$ ;

4) Быстро степенуване  $\rightarrow \Theta(\log(n))$

if ( $x == 0$ )

return 1;

if ( $x \% 2 == 0$ )

return  $n * \text{power}(n * n, x/2)$ ;

else {

return  $n * \text{power}(n, x-1)$ ;

5) Сума на ел. в масив

if ( $arr == \text{nullptr} \text{ || } size == 0$ )

return 0;

return  $arr[0] + \text{sum}(arr+1, size-1)$ ;



6) max el & maxB  $\rightarrow \Theta(n)$

return  $\max(\text{arr}[0], \max(\text{arr}, (\text{arr}+1, \text{size}-1)))$

7) Linear Search

return  $\text{arr}[0] == x \text{ || linearSearch}(\text{arr}+1, \text{size}-1, x)$

8) Binary Search

```
bool BinarySearch(---) {
    int mid = left + (right - left) / 2;
    if (mid == el) return true;
    if (mid > el) arr[mid];
    return BinarySearch(left,
                        right, el);
    else
        return BinarySearch(left,
                            arr[mid],
                            size - mid - 1, x);
```

9) Palindrome

```
bool isPalindrome (const
    char* str, size_t len) {
```

len <= 0 || len == 1)

return true;

return  $\text{str}[0] == \text{str}[\text{len}-1]$  &  
isPalindrome(str+1, len-2)

10) Брой големи и малки букви

if ( $\text{str} == '\backslash 0'$ )  $\text{lows} = \text{upps} = 0$ ;  
return;

if ( $\text{isLower}(\text{str}[0])$ )

$\text{lows}++$ ;

if ( $\text{isCapital}(\text{str}[0])$ )

$\text{upps}++$ ;

get how and upps count ( $\text{str} + 1$ ,  
lows, upps);

no preferencija