ХИМИКОТЕХНОЛОГИЧЕН И МЕТАЛУРГИЧЕН УНИВЕРСИТЕТ – СОФИЯ

ИНФОРМАТИКА

част втора

Основи на програмирането на C++

лектор: гл. ас. д-р Стефан М. Панов

Катедра “Информатика”

Лекция 8

Обработка на низове в C++. Входно-изходни манипулатори. Изброявания.

Библиотечни функции за обработка на низове

Езикът C++ поддържа множество функции за обработка на низове. Ще разгледаме най-употребяваните от тях. Заглавният файл, който е необходимо да се включи към програмата, за да се извикват тия функции е **<cstring>**
(или <string.h>).

Извикването на функцията **strlen(str)** връща дължината (броя на символите) на **str**, зададен като неин аргумент. Нулевият символ не се отчита. **str** може да бъде както **масив от символи** (в който се съхранява низ), така и **низова константа**. **Забележете,** че когато е **str** масив, функцията **strlen** връща дължината на низа, който се съхранява в масива, а не размера на масива.

За следващите две функции **първият аргумент е масив от символи**, а вторият аргумент може да бъде както **масив от символи** (в който се съхранява низ), така и **низова константа**.

Форматът за извикване на функцията strcpy () е: **strcpy (to, from);** **Действие:**

Съдържанието на низа **from** се копира в масива **to**. Масивът **to** трябва да е достатъчно голям, така че в него да се помести низа от **from**. В противен случай масивът **to** ще се препълни, т.е. ще се излезе извън границите му, което може да доведе до срив на програмата.

Форматът за извикване на функцията strcat () е: **strcat (s1, s2);** **Действие:**

Копие на низа **s2** се добавя към низа в масива **s1**. При това действие низът **s2** не се променя. Първият символ на **s2** се записва на мястото на нулевия символ на низа в **s1**. Лесно се съобразява, че размерът на масива **s1**, трябва да е поне **(strlen(s1) + strlen(s2) + 1)** байта, за да се гарантира, че ще може да побере всички символи на низа **s2** плюс нулев символ. Ако това изискване не е изпълнено поведението на функцията е неопределено.

За двата аргумента на следващата функция **strcmp()** е в сила казаното за аргумента на функция **strlen()**. Форматът за извикване на функцията е: **strcmp (s1, s2);** **Действие:** сравняват се низовете **s1** и **s2**. Низовете се сравняват символ по символ, като се взимат числовите стойности на символите. **Сравняването спира и се връща резултат, когато се срещнат различни символи или нулев символ в един от двата низа.**

Функцията **strcmp()** връща **0**, ако низовете са еднакви, **1** (строго казано положително число), ако **s1** е по-голям от **s2** (текущият символ в **s1** има по-голяма числова стойност от текущият символ в **s2** или символът в **s2** е нулев символ), **-1** (строго казано отрицателно число), ако **s1** е по-малък от **s2** (символът в **s1** има по-малка числова стойност от символа в **s2** или символът в **s1** е нулев символ).

Функцията **strstr()** ще се разгледа, когато се запознаем с указателите.

В следващата програма_35 са използвани и 4-те разгледани функции.

```
#include <iostream> // nashata 35-a programa
#include <cstring> // obrabotka na nizove
using namespace std;
int main() {
	char s1[80], s2[80]; // dva simvolni masiva
	cout << "Vavedete tekst: ";
	gets (s1);
	cout << "Vavedete drug tekst: ";
	gets(s2);
	cout << "\nDaljinata na pyrvia text e: " << strlen(s1);
	cout << " a na vtoria:" << strlen(s2) << '\n';
	if(!strcmp(s1, s2)){ // zabelejete otricanieto !
		cout << "Tekstovete sa ednakvi!\n";
	}
	else {
		cout << "Tekstovete sa razlichni!\n";
	}
```

```
    cout << "\nDa gi sravnim pak sled obrabotka.\n" << '\n';
    strcat(s1, " ");
    strcat(s1, s2);
    strcpy(s2, s1);
    cout << s1 << " ? " << s2 << ' ';
    cout << "\n\nSega sa ednakvi!\n";
    return 0;
}
```

Забележете, че функция **strcmp()** връща стойност **0** (**false**), ако низовете са равни. Ето защо, ако проверяваме два низа за равенство е необходимо да използваме оператор “! ” (НЕ), за да обърнем стойността на **true**, както и е направено в програмата.

Библиотечни функции за преобразуване на низ в число

Стандартната библиотека на C++ включва и три функции, които позволяват преобразуване на символен низ, представляващ десетична числова стойност в числовия му еквивалент. Функциите **atoi()** и **atol()** преобразуват низа в целочислена стойност съответно с тип **int** и **long int**, докато **atof()** преобразува в реално число от типа **double**. За да използваме тия функции е необходимо да включим в програмата заглавния файл **<cstdlib>**.

Трите функции работят подобно: Всички водещи „бели-символи“ се пренебрегват. Бели-символи са интервал ' ', хор. табулация '\t', вер. табулация '\v', връщане на курсора '\r', нов ред '\n', нова страница '\f' (важните в **синьо**). Ако в низа се срещне непозволен символ, той и всички символи след него се пренебрегват и функцията преобразува в числов еквивалент само тая част от низа, която се намира вляво от непозволения символ. За функцията **atof()** са разрешени и двата вида запис на реалното

число – с фиксирана и плаваща точка, примерно **"-7935.5678"** и **"-7.9355678E3"**. **Ако низът не може да се преобразува в число и трите функции връщат 0.**

Програма_36 е пример за такива преобразувания. Следват разяснения относно нейния код и резултатите, получени при изпълнението ѝ. При второто преобразуване всички символи преди числото са само бели, затова те се игнорират и преобразуването е успешно. Но когато се извежда низа str на екран управляващият символ "\n" премества курсора на нов ред и чак тогава се отпечатва числото.

При третото извеждане низът "-2345" се изписва за кратко на екрана, но след това управляващият символ "\r" премества курсора в началото на текущия ред, **което води до изтриването на низа**. След това маркерът се премества един интервал надясно заради интервала в низа след буквата "r". Това обяснява защо "se" започва от трета позиция на третото извеждане.

```
#include <iostream> // nashata 36-a programa
#include <cstring> // preobrazuvane na  nizove v chisla
#include <cstdlib> // zaradi atoi(), atol() i atof()
#include <iomanip>
using namespace std;
int main() {
      char  str[32]; int i; // za atoi()
      long l; // za atol()
      double d; // za atof()
      i =  atoi("-2345"); // pyrvo preobrazuvane
      cout << "-2345" << " se preobrazuva v " << i << "\n";
      strcpy (str, "  \t\n\r-2345 ");
      i =  atoi(str); // vtoro preobrazuvane
      cout <<   str  << " se preobrazuva v " << i << "\n";
      strcpy (str, "  \t\n-2345\r ");
      i =  atoi(str); // treto preobrazuvane
```

```
cout <<   str  << " se preobrazuva v " << i << "\n";
strcpy (str, "  \t\n\r-2300000000.45"); // ! tova chislo NE se sybira v 4 baita
l =  atol(str); // chetvyrto preobrazuvane, vzimat se mladshite 4 baita
cout << str << " se preobrazuva v " << l << "\n";
strcpy (str, "  \t\n\r-230000000.45"); // ! tova chislo se sybira v 4 baita
l =  atol(str); // peto preobrazuvane
cout << str << " se preobrazuva v " << l << "\n";
strcpy (str, "   \t$\r-23.45"); // \r shte otide v nachaloto na reda i taka $ se iztriva
l = atol(str); // shesto preobrazuvane
cout << str << " se preobrazuva v " << l << "\n";
strcpy (str, "  \t$\n\r-23.45"); l = atol(str); // sedmo preobrazuvane
cout << str << " se preobrazuva v " << l << "\n";
strcpy (str,   "-7935.5678"); d = atof(str); // osmo preobrazuvane
cout << str << " se preobrazuva v   " << d << "\n";
cout << setprecision(8)<<setiosflags(ios::fixed)<< str<<" se preobrazuva v   "<<d << "\n";
strcpy (str,   "-7.9355678E3");d = atof(str); // deveto preobrazuvane
```

```
    cout << str << " se preobrazuva v " << d << "\n";
    return 0; }
```

Четвъртото преобразуване е най-сложното. Числото -2300000000 не се събира в 4 байта, т.е. 32 бита, най-левият от които е за знака плюс или минус. Двоичното представяне на това число **в допълнителен код** в 8 байта е:

11111111 11111111 11111111 11111111 01110110 11101000 11001001 00000000

Функцията atol() обаче работи с типа long затова се взимат само младшите 4 байта (дадени в червено). Най-левият бит е 0, т.е. това е положително число. Преобразувано в десетична БС това е числото **1994967296**, което виждаме на екрана. Точката е недопустим символ при целите числа, затова тя и двете цифри след нея се пренебрегват.

Заслужава да се отбележи, че за нашият компилатор и типът int, и типът long int заемат по 4 байта. Което води до заключението, че при нас функциите atoi и atol работят еднакво!

Петото преобразуване води до число, което се събира в 4 байта, затова липсват проблемите от предходния случай.

При шестото и седмото преобразуване недопустимият символ $ е в началото на низа затова atol(str) връща 0. Заради проблемите, вече описани с управляващия символ "\r" при шестото извеждане самият символ $ не се вижда, но при седмото преобразуване е изведен след хоризонталната табулация. При осмото и деветото извеждане е показано, че се получава еднакъв резултат и при двете представяния – с фиксирана и плаваща точка - на реалното число. При настройките по подразбиране се печатят общо 6 цифри и то след като предварително е направено закръглението. Когато обаче с помощта на инструкциите **setprecision(8)** и **setiosflags(ios::fixed)**

укажем да се отпечатват 8 цифри след десетичната точка се вижда, че началният низ точно е преобразуван в съответното десетично число.

Да се върнем към програма_32 от предишната лекция. Там, ако въведем примерно реалното число **12.34** при изпълнение на инструкцията **cin >> i,** която очаква цяло число заради типа **int** на променливата **i** ще се влезе във вечен цикъл. **Затова по-доброто решение е да се въвежда в низ , а той да се преобразува в число с функцията atoi()**. Т.е. инструкцията **cin >> i;** да се замени с тия две: **gets(str); i = atoi(str);** В новия вариант програмата ще работи без срив, каквото и да въведем от клавиатура в масива **str**.

Масив от редове (от низове)

За създаване на масив от редове се използва двумерен символен масив, в който размерът на левия индекс определя броя на редовете, а размерът на десния — максималната дължина на всеки ред. Достъпът до даден ред е лесен: достатъчно е да се укаже само левият индекс.

Следващата програма_37 чете текст от клавиатура по редове и и го показва на екрана. Само при въвеждане на празен ред **text[t][0]** е **0**, защото в началния байт на текущия ред **t** се записва нулевият символ **'\0'** за край на низа.

```
#include <iostream> // nashata 37-a programa
#include <cstring> // vavejdame redove s tekst ot klaviatura
using namespace std; // i gi pokazvame na ekrana
int main() {
	int t, i;
```

```
char text[20][80];
for(t=0; t<20; t++) {
      cout<< t<< ":" ;
      gets(text[t]);
      if (!text[t] [0]) {
            break; // izhod ot cikyla pri vavejdane na prazen red
      };
}
for(i=0; i<t; i++) { // izvejdane na redovete na ekrana
      cout << text[i] << "\n";
}
return 0;}
```

Функции за обработка на символи

Освен функции за обработка на низове в езика C++ са предвидени и множество функции за обработка на символи. От тях подмножеството функции, чийто имена започват със сричката „is“ работят по сходен начин: **приемат за аргумент символ и връщат стойност, различна от 0, ако дадено условие е изпълнено, в противен случай връщат 0**.

Ето част от тия функции с показно за всяка от тях условие:

isalnum: символът е буква A÷Z или a÷z или цифра 0÷9
isalpha: символът е буква A÷Z или a÷z
isdigit: символът е цифра 0÷9
islower: символът е малка буква a÷z
ispunct: всеки печатен символ без интервал, a÷z, A÷Z и 0÷9
isspace: символът в бял (белите символи са вече разгледани)
isupper: символът е главна буква A÷Z

Освен изброените функции има и други, от които ще разгледаме две:

toupper : ако аргументът е малка буква от a÷z връща съответната главна;
tolower : ако аргументът е главна буква от A÷Z връща съответната малка;
Ако условието не е изпълнено и двете функции връщат аргумента без изменение.

Следващата програма_38 показва употребата на повечето от разгледаните функции. За всяка от включените „**is**“ функции е предвиден брояч, който се увеличава с 1 винаги когато „неговата“ функция върне стойност, различна от 0. Проверката се прави върху 3 реда с текст.
Накрая в първия ред малките букви се сменят с главни. Обърнете внимание как се реализира управляващ-израз на последния **for**. Да си припомним, че цикълът **for** завършва, когато управляващ-израз стане 0. Но **text[0][i]** е ASCII кода на „**i**“-я символ. Той ще стане 0 когато е достигнат края на низа в **text[0], защото краят на всеки низ се задава с нулевия символ. Т.е. цикълът ще завърши точно когато трябва, когато низът е обработен докрай.**

```
#include <iostream> // nashata 38-a programa
#include <cstring> // funkcii za obrabotka na simvoli
using namespace std;
int main() {
	int t, i; char text[3][80]; // dvumeren masiw ot redowe
	int broiach_na_bukvi = 0; // kakto i sumite, broiachite
	int broiach_na_cifri = 0; // syshto se nulirat v nachaloto
	int broiach_na_malki_bukvi = 0;
	int broiach_na_glavni_bukvi = 0;
	int broiach_na_punct_simvoli = 0;
	int broiach_na_beli_simvoli = 0;
	strcpy (text[0],"1. Edin tekst, \tna koito shte smenim bukwite w glawni.\n");
	strcpy (text[1],"2. Drug tekst, \t 2-3 drugi dumi!");
	strcpy (text[2],"3. Treti tekst (), dali ni triabva?\n");
	for(t=0; t<3; t++) { // izvejdane na redovete na ekrana
		cout << text[t] << "\n";
	}
```

```
    for(t=0; t<3; t++) { //  za vseki red
        for (i=0;i<strlen(text[t]); i++){ // za vseki simvol v reda pravi 6 proverki
            if (isalpha(text[t][i])) {broiach_na_bukvi++;}
            if (isdigit(text[t][i])) {broiach_na_cifri++;}
            if (islower(text[t][i])) {broiach_na_malki_bukvi++;}
            if (isupper(text[t][i])) {broiach_na_glavni_bukvi++;}
            if (ispunct(text[t][i])) {broiach_na_punct_simvoli++;}
            if (isspace(text[t][i])) {broiach_na_beli_simvoli++;}
        } } //
    cout << "V teksta ima:\n\n" << broiach_na_bukvi << " bukvi\n"
<< broiach_na_cifri << " cifri\n" << broiach_na_malki_bukvi << " malki bukvi\n"
        << broiach_na_glavni_bukvi << " glavni bukvi\n"
        << broiach_na_punct_simvoli << " simvola za punctoacia\n"
        << broiach_na_beli_simvoli << " beli simvoli\n" ;
    for (i=0; text[0][i]; i++) { text[0][i] = toupper(text[0][i]); }
    cout << "\nA sega pyrviat red e s glavni bukvi:\n\n"<< text[0] << "\n";
    return 0;}
```

Входно-изходни манипулатори

Досега при въвеждане или извеждане на информация в разгледаните примери действаха параметрите за форматиране, които се използват по подразбиране във входно-изходната система на C++. Но програмистът може сам да управлява формата на представяне на данните, при това по два начина. Първият от тях предполага използването функциите-членове на класа **ios**, а вторият — функции от специален тип, наречени манипулатори. Ще се спрем на втория подход.

Манипулаторите позволява да вграждаме инструкции за форматиране във входно-изходните изрази.

Някои от по-рядко използваните манипулатори и флагове няма да бъдат разглеждани.

При използване на манипулатори, които приемат аргументи е необходимо да включим в програмата заглавния файл <iomanip>.

Една особеност на манипулаторите – ако даден манипулатор се извиква без аргументи (например, **endl**), то неговото име се указва без кръгли скоби.

setw() – изходен манипулатор. Извиква се така: **setw(целочислен_израз)**. **Задава ширината на полето за следващото извеждане** да бъде равна на стойността на целочисления израз. И числовите, и текстови данни се подреждат отдясно на полето.

setprecision(**целочислен_израз**) – изходен манипулатор. Стойността на израза задава броя на **всичките** позиции на **реално** число, т.е. и за цялата и за дробната част общо. **Манипулаторът е в сила за всичките извеждания на числа след него**.

setiosflags(**ios::fixed**) – когато тоя манипулатор с указания параметър **ios::fixed** се използва заедно със **setprecision** целочисленият израз задава само броя на позициите за **дробната** част на реално число. **setiosflags** може да се използва и с други параметри, (всички негови параметри се наричат

флагове) един от които е **ios::scientific –** реалното число ще се извежда с мантиса и порядък, т.е. във **формат плаваща точка**.

resetiosflags(флаг) – нулира флага, зададен като параметър в **setiosflags**. **Важно**: За да зададем нова стойност на флаг, първо трябва да го нулираме. Иначе извеждането ще бъде както са настройките по подразбиране. Трябва да знаем и флаговете!

setfill(символ) - изходен манипулатор. Задава символ запълнител когато полето зададено със **setw** е по-дълго от извежданата стойност.

oct - изходен манипулатор. **Всички** следващи извеждания на **цели числа** ще бъдат в осмична бройна система (БС). Аналогично се използват **hex** и **dec** за 16-ична и 10-ична БС. Вместо трите може да се ползват съответно **setbase(8)**, **setbase(16)**, **setbase(10)**.

ws – **входен**,пропуска входните интервали при **следващото** въвеждане на низ.

endl – манипулатор без параметри. Извежда **"\n"** – символа за нов ред. Ето защо **може да се ползва вместо "\n", но никога не се поставя вътре в низ, а винаги като отделен елемент на инструкция cout.** Също така изпразва буфера, т.е. съдържанието на буфера, свързан с изходния поток се записва на изходното устройство.

flush – манипулатор без параметри. Също изпразва съдържанието на буфера.

Внимание: Редуването на **cin** и **gets()** може да доведе до проблеми при въвеждане. В такъв случай между двете инструкции трябва да се постави **cin.sync()**; **Инструкцията изчиства входния буфер!**

Следващата програма_39 показва как се ползват разгледаните манипулатори.

```
#include <iostream> // nashata 39-a programa
#include <cstring> // primerno izpolzvane na slednite manipulatori
#include <iomanip> // endl, setw, setprecision, setiosflags
using namespace std; // resetiosflags, setfill, hex, oct, dec, ws
int main() { char text[80];
      float a = 12345.4567;
      double b = 12345.4567;
      int m = 1234567; // !! ako komentirame resetiosflags ne raboti pravilno.
// t.e. pyrvo triabva da nulirame flag za da zadadem drug
// inache (pri komentirane) raboti po podrazbirane
cout << "a="<< a<<endl<< setprecision (6) << setiosflags(ios::fixed)<<"a="<<a<<
endl << resetiosflags(ios::fixed) <<   setiosflags(ios::scientific)<< "a="<< a<< endl ;
// zabelejete razlikata w tochnostta mejdu tip float i double
// zashto s flag left se dyrji razlichno: podravniava otliavo i zapylwa otdiasno
```

```
// zabelejete, che hex ne vazdeistva na promenlivata b, zashtoto ne e tip int
      cout << resetiosflags(ios::scientific) << "b="<< b<< endl<< setprecision (6) <<
      setiosflags(ios::fixed) << "b=" << /* setiosflags(ios::left) << */ hex <<
      setw(16)<<b<<endl<<"b="<<setw(16)<< setfill('$')<<b<<endl << "m(16)="<<
      m<<endl<<oct<<"m(8) ="<<m<<endl<<dec<< "m(10)="<<m<<endl
      <<resetiosflags(ios::fixed)<<setiosflags(ios::scientific)<<"b="<<b<< endl;
// !!! wywedete "45" i "45 da" s i bez cin.sync();
      cout << "vavedete chislo: ";  cin >> m;
      cin.sync(); // nulira whodnia bufer
      cout << "m=" << m << endl << "vavedete tekst s vodeshti intervali\n";
      gets(text) ; // vodeshtite intervali sa tuk
      cout << "text=" << text << endl;
      cout << "vavedete tekst s vodeshti intervali\n";
      cin >> ws ; // ws vaji samo pri sledvashtoto vavevdane
      gets(text); // cin >> ws; triabva da e tochno predi gets
      cout << "text=" << text << endl; // bez vodeshtite intervali
  cout << "vavedete tekst s vodeshti intervali\n"; gets(text);// tuk ws veche ne vaji
```

```
    cout << "text=" << text << endl;// vodeshtite intervali sa tuk
    return 0;
}
```

Пояснения: употребата на някои от манипулаторите крие някои тънкости, които ще бъдат разяснени:

При извеждането на променливата **a** не можем да преминем към представяне с мантиса и порядък, ако не извикаме преди това манипулатора resetiosflags(ios::fixed). Другото, което се забелязва е загубата на точност в дробната част на числото, когато за нея са заделени 6 цифри. Причината е, че битовете заделени за мантисата в едно 4 байтово число са недостатъчни за представянето на началното число 12345.4567. Но при осембайтовото представяне на същата стойност в променливата b мантисата е достатъчно дълга и числото се представя точно. Манипулаторът setiosflags(ios::left) показва използването на друг флаг (left) след който стойностите се подравняват отляво, когато ширината на полето е зададена

със setw и е по-голяма от дължината на самото число. Запълващият символ по подразбиране е интервал, но може да го променим със setfill. Умишлено в програмата трите извеждания на променливата m за трите бройни системи са преди последното отпечатване на b. Намерението е да се покаже, че **манипулаторите hex, oct и dec не влияят върху извеждането на реални числа**.

Втората част на програмата показва ролята на **cin.sync()**. Без тая инструкция информацията от входния буфер, въведена за променливата **m** се използва за вход в инструкцията gets(text). По-точно казано не цялата, а това което се намира след цифрите на **m**. Т.е. дори ако след цифрите няма нищо стойността на text след изпълнение на gets(text) ще бъде един празен низ (""). Вижда се също, че входният манипулатор **ws** работи както е описано.

Изброявания

Изброяването е средство, чрез което могат да се създават именувани целочислени константи. Определянето на изброяване става по следния начин:

```
enum име-изброяване
{
  име-константа1,
  име-константа2,
  ...
  име-константаN
};
```

Дефиницията на едно изброяване е видима от точката на деклариране до края на блока (ако е във функция)или файла, в който се намира дефиницията. **По-подразбиране** **име-константа1** се инициализира автоматично с **0**, **име-константа2** с **1** и т.н., тоест всяка следваща константа от

списъка е с единица по-голяма от предходната. Ще наричаме тия константи **enum-константи**. Enum-константите могат изрично да бъдат инициализирани от потребителя със следната форма:

```
enum име-изброяване
{
  име-константа1 = инициализатор1,
  име-константа2 = инициализатор2,
...
  име-константаN = инициализаторN
};
```

Където **инициализатор1, ... , инициализаторN** може да бъде всеки целочислен константен израз от тип **int**.

Важно е да се запомни, че всяка константа, която не е изрично инициализирана, е с единица по-голяма от предходната. Типът на enum-константите технически съвпада с тип int, т.е. инициализаторът може да

има стойности, каквито типът int може да съдържа за съответната система. Друго, което трябва да се знае е, че **горните две конструкции не водят до заделяне на памет.** Това са конструкции, чрез които само задаваме имена на целочислени константи.

Всяко определяне на изброяване създава нов тип, т.е. можем да дефинираме и променливи от типа **enum**. На enum-променливата могат да се присвояват стойности от списъка с enum-константи. По същество enum-променливите са целочислени променливи. Действителният тип, който стои зад един enum-тип зависи от компилатора. Компилаторът може да избере най-подходящия от целочислените типове **char** (signed, unsigned), **short** (signed, unsigned) или **int** в зависимост от стойностите на enum-константите.

Следва проста програма_40 с използването на изброяване. Обърнете още един път внимание, че константите са цели числа и ако искаме да видим подходящ текст се налага сами да се погрижим. В нашия пример това е

направено с помощта на инструкция **switch**. Струва си да се отбележи, че не можем да въведем стойност на променлива от тип изброяване от клавиатурата защото инструкция **cin** е имплементирана само за стандартните типове данни.

```
#include <iostream> // nashata 40-a programa
#include <cstring> // primer za nov tip izbroiavane
using namespace std; // ako po niakakvi prichini iskame da promenim
enum color {black, brown, blue=8, green} ;// stoinostta  primerno na blue
int main() {
	color ochi; // moje i enum color ochi;
	int eyes;
// ochakva se, che raboteshtiat s programata znae nashite enum konstanti
	cout << "Molia vavedete cveta na ochite: ";
	cin >> eyes;
// !!! Ne moje    cin >> ochi , cin e samo za standartnite tipove
	ochi = (color) eyes; // iavno preobrazuvane na tipa
```

```
    cout << "\nIzbrahte ";
    switch (ochi) {
        case black:cout << "cherni";
            break;
        case brown:cout << "kafiavi";
            break;
        case blue:cout << "sini";
            break;
        case green:cout << "zeleni";
            break;
        default: cout << "s neizvesten cviat";
    }
    cout << " ochi!\n";
    return 0;
}
```