Reťazce

Reťazec je postupnosť znakov uložených za sebou v pamäti. Používame dva spôsoby práce s reťazcami. Prvým, je spôsob, ktorý bol prevzatý od jazyka C a naučíme sa ho teraz. Druhý, je využitie knižnice **cstring**, a to si ukážeme v neskorších pokračovaniach. Zatiaľ nám postupnosť znakov uložených za sebou v pamäti pripomína pole, a je to správna úvaha. Reťazec je pole prvkov typu char, kde má každý znak svoj prvok poľa. Reťazce ponúkajú vhodný spôsob ukladania textových informácií. Reťazce majú ešte jednu zvláštnu vlastnosť, posledný znak každého reťazca je nulový znak. Tento znak má zápis **'\0'** a má kódovú hodnotu ASCII 0, slúži na ukončenie reťazca. Pozrite sa na nasledujúce dve deklarácie:

char pes[5] = {'p', 'e', 't', 'k', 'o'}; //nie je retazec

char macka[5] = {'c', 'i', 'c', 'a', '\0'}; //retazec

Obidve polia sú typu char, ale iba druhé je reťazcom. Jazyk C++ ponúka funkciu pre prácu s reťazcami, ktorá používa objekt cout. **Všetky funkcie spracúvajú reťazce, až kým nenarazia na nulový znak**. Ak použijeme cout, aby vypísal premennú mačka, tak vypíše cica, prečíta nulový znak a skončí. Ale ak mu dáme vypísať premennú pes a on vypíše pole znakov, ktoré nasleduje v pamäti až kým niekde v pamäti nenájde nulový znak. Aj keď sa tento znak v pamäti vyskytuje celkom často nemôžeme takéto pole brať za reťazec. Podľa príkladu poľa mačka vyzerá inicializácia dosť zložito. Ale existuje aj jednoduchší spôsob inicializácie reťazcov. Stačí jednoducho použiť reťazec v úvodzovkách, ktorý sa nazýva **reťazcová konštanta**. Napr.:

char meno[10] = "Pan Peter"; //'\0' sa doplni samo ale musite mu tam nechat miesto

char jav[] = "Bublinky"; //Velkost si spocita prekladac a doplni si sam '\0'

Reťazcom v úvodzovkách je vždy doplnený nulový znak, tak ho nemusíte vypisovať. Taktiež aj **vstupy z klávesnice si samo dopĺňajú nulový znak**. Ale musíme si byť istý, že **pole je dostatočne veľké pre uloženie všetkých znakov, vrátane nulového znaku**. Inicializácia znakového poľa reťazcovou konštantou je jeden z prípadov, kedy môže byť bezpečnejšie nechať určenie počtu prvkov na prekladači. Ak vytvoríte pole väčšie než je reťazec, okrem zaberaniu zbytočne pamäte neurobíte nič zlé. Je to preto, lebo funkcie pracujúce s reťazcami, neriadia veľkosť poľa ale hľadajú nulový znak. Všimnite si, že reťazcová konštanta je nezameniteľná za znakovú konštantu. Znaková konštanta je „S“ predstavuje zápis kódu znaku. Na systéme so znakovou sadou ASCII je 'S' iba iným zápisom čísla 83. Preto príkaz:

char shirt_size = 'S'; //toto je v poriadku

priradzujeme hodnotu 83 premennej shirt_size. Ale **reťazcová konštanta „S“ reprezentuje reťazec zložený z dvoch znakov S a '\0'.** Horšie je, že v skutočnosti predstavuje adresu pamäte, na ktorej je reťazec uložený. Takže príkaz:

chart_size = "S"; // nepovolene miesanie typov

sa pokúša premennej chart_size priradiť pamäťovú adresu. Ale v C++ má adresa samostatný typ a preto to prekladač nedopustí.

Spájanie reťazcov

C++ umožňuje spájanie reťazcových konštánt, to znamená spojenie dvoch reťazcov do jedného. Ľubovoľné dva reťazce oddelené iba bielymi znakmi ako medzera, tabulátor alebo znakom nového riadku. Všetky nasledujúce príkazy sú rovnaké:

```
cout << "Som programator ale musim sa este " "dost ucit aby som bol dobry.\n";
cout << "Som programator ale musim sa este dost ucit aby som bol dobry.\n";
cout << "Som programator ale musim sa este "
"dost ucit aby som bol dobry.\n";
```

Všimnite si, že spojenie nepridáva žiadne medzery do spojených reťazcov. Prvý znak druhého reťazca bezprostredne nasleduje za posledným znakom prvého reťazca, čiže znak „\\0“ sa nepoužíva.

Použitie reťazcov v poliach

Dva najbežnejšie spôsoby vkladania reťazcov do poľa, sú inicializácie poľa reťazovou konštantnou a načítanie vstupu do poľa z klávesnice alebo zo súboru. Teraz si to ukážeme za použitia objektu cin. Program ďalej používa knižnú **fukncіu strlen()** na zistenie dĺžky reťazca, kvôli tej to funkcii sme pridali do súboru knižnicu **cstring**, ktorá nám to dovolí využívať a obsahuje aj ďalšie funkcie na prácu z reťazcami.

```
#include <iostream>
#include  <cstring>// pre funkciu strlen()

using namespace std;

int main(){
        const int velkost=20 ;
        char meno1[velkost];  //prazdne pole
        char meno2[velkost]= "ProgramerC++"; //inicializacia pola

        cout << "Dobry den! Ja som " << meno2 << "\n";
        cout << "Kto ste vy??\n";
        cin >> meno1;
        cout << "Dobre, "<< meno1 << " vase meno obsahuje ";
        cout << strlen(meno1) << " znakov a je ulozene\n";
        cout << "v poli o velkosti "<< sizeof(meno1) << " bajtov.\n";
        cout << "Prve pismeno vasho mena je "<< meno1[0] << ".\n";
        meno2[9]='\0'; // nulovy znak
        cout << "A tu je prvych 9 znkov mojho mena: "<< meno2;
        cout << "\n";
        return 0;
}
Dobry den! Ja som ProgramerC++
Kto ste vy??
Martin
Dobre, Martin vase meno obsahuje 6 znakov a je ulozene
v poli o velkosti 20 bajtov.
Prve pismeno vasho mena je M.
A tu je prvych 9 znkov mojho mena: Programer
```

Najprv si všimnite, že operátor sizeof vracia veľkosť celého poľa 20 bajtov ale funkcia strlen() vracia iba dĺžku reťazca a nie celého poľa. **Funkcia strlen() vracia veľkosť viditeľných znakov bez nulového**. Preto vracia hodnotu 6 namiesto 7. Ale na jeho uloženie potrebujeme pole o veľkosti

strlen(meno1)+1. Pretože meno1 a meno2 sú polia, dá sa k ním pristupovať pomocou indexov. To využíva náš program, keď prečíta iba prvý znak reťazca meno1[0]. Ďalej sme nastavili meno2[9]. A to nám spôsobili **skrátenie reťazca na 9 znakov aj keď ďalšie znaky v poli ostávajú**. Nakoniec si ešte všimnite, že na určenie veľkosti poľa sme použili symbolickú konštantu, to nám zjednodušuje úpravu programu pri zmenách veľkosti programu. A ešte pozor, ak do programu zadáte viac ako 20 znakov (19+nulový), tak Vás operačný systém zastaví, že chcete využívať pamäť, ktorú nemáte blokovanú.

Problémy so vstupom

Program má chybu, ktorú si teraz opravíme a vysvetlíme. Ale na začiatok ešte jeden príklad, ako by sa to nemalo robiť:

```
// retazce2.cc -- citanie viacerych retazcov

#include

using namespace std;

int main() {
        const int velkost=20;
        char meno[velkost];
        char dezert[velkost];

        cout << "Zadajte vase meno: \n";
        cin >> meno;
        cout << "Zadajte vas oblubeny dezert: \n";
        cin >> dezert;
        cout << "Mam bohovsky dezert "<< dezert;
        cout << ", ktory si zasluzi iba ten "<< meno << ".\n";
        return 0;
}
Zadajte vase meno:
Martin Bodocky
Zadajte vas oblubeny dezert:
Mam bohovsky dezert Bodocky, ktory si zasluzi iba ten Martin.
```

No, a ako sme zistili, nebolo nám ani umožnené zadať názov dezertu. Program síce vypísal požiadavku na zadanie dezertu, ale hneď za ním vypísal poslednú správu. Problém spočíva v tom, že **objekt cin rozoznáva koniec reťazca**. A z klávesnice sa nedá zadať nulový znak, tak objekt cin potrebuje iný spôsob ukončenia reťazca. **Objekt cin preto používa pre určenie nulového znaku medzery, tabulátory a znaky nového znaku**. Preto objekt cin načítal iba prvý reťazec nášho vstupu a ukončí ho nulovým znakom. V praxi to znamená, že cin prečíta Martin ako prvý reťazec a umiestni ho do poľa meno. Reťazec ostane vo vstupnej fronte. Potom, keď hľadá cin odpoveď na druhý reťazec, nájde Bodocky a vloží ho do poľa dezert. Ďalší problém, ktorý sme si ešte nedemonštrovali, a to je to, že nám program nezabráni vložiť dlhší reťazec ako 20 znakov.

Riadkovo orientovaný vstup: getline() a get()

Aby sme mohli, ako reťazec, ukladať celé vety a nemuseli by sme čítať vstup po slovách potrebujeme iný prístup. Treba použiť riadkovú orientáciu. Našťastie máme knižnicu iostream, ktorá obsahuje objekt cin a ten ma ešte funkcie **cin.getline() a cin.get(). Cin.getline() prečíta celý riadok a pre označenie konca používa znak nového riadku respektíve ENTER. Preberá dva argumenty, a to meno poľa pre uloženie vstupného riadku, druhý argument obmedzuje počet vstupných znakov**. Ak je to napríklad 20, tak prečíta 19 znakov a ponechá miesto na nulový znak. čiže **funkcia cin.getline končí ak prečíta požadovaný počet znakov, alebo narazí na znak konca riadku**. Volanie funkcie vyzerá tak to:

```
cin.getline(meno,20);
```

Tento príkaz prečíta celý riadok do poľa meno o maximálnej dĺžke 19 znakov plus nulový znak. Tak si ukážeme opravenú verziu nášho programu:

```
// retazce3.cc -- citanie viac ako jedneho slova pomocou getline
#include
using namespace std;
int main() {
        const int velkost=20;
        char meno[velkost];
        char dezert[velkost];

        cout << "Zadajte vase meno: \n";
        cin.getline(meno,velkost); //precita cely riadok
        cout << "Zadajte vas oblubeny dezert: \n";
        cin.getline(dezert,velkost);
        cout << "Mam vyborny "<< dezert;
        cout << ", ktory si zasluzi iba " << meno << ".\n";
        return 0;
}
```

Program teraz prečíta meno aj dezert a vypíše ho podľa očakávania. Prečíta vstup až do znaku nového riadku, ktorý označuje jeho koniec. Ale teraz skúsime iný prístup, lebo poznáme ešte druhú funkciu na orientovaný vstup a to **cin.get(),** ktorá má niekoľko variant. Jedna z nich pracuje pomaly, tak isto ako cin.getline(). **Preberá rovnaké argumenty a číta tiež do konca riadku. Ale namiesto prečítania znaku konca riadku ho nechá vo vstupnej fronte**. Predpokladajme použitie dvoch funkcii za sebou:

```
cin.get(meno,velkost);
cin.get(dezert,velkost);  // tu vznika problem
```

Pretože prvé volanie nechá znak konca riadku vo vstupnej fronte a tento znak je prvý, ktorý uvidí druhá funkcia cin.get(). Preto si táto funkcia myslí, že už dosiahla konca aj bez toho, že by niečo prečítala. Bez nejakej pomoci sa funkcia cin.get() cez tento znak nedostane. Ale môžeme si pomôcť volaním funkcie cin.get() bez parametrov, ktorá prečíta jeden znak aj keď je to znak konca riadku. To môžeme použiť pri odstránení znaku konca riadku a prípravu na ďalší vstup. napr.:

```
cin.get(meno,velkost);  //precita prvy riadok
cin.get();  //precita znak konca riadku
cin.get(dezert,velkost);  // precita druhy riadok
```

Ďalšou možnosťou použitia funkcie cin.get() je spojenie dvoch členov knižnice:

```
cin.get(meno,velkost).get();
```
Tento zápis umožňuje skutočnosť, že funkcia cin.get(meno,velkost) vracia objekt, ktorého použitie môže vyzerať aj takto:

```
cin.getline(meno1,velkost).getline(meno2,velkost);
```
ktorý prečíta dva za sebou idúce riadky do poli meno1 a meno2, presne ako dve za sebou idúce funkcie cin.getline(). Teraz budeme demonštrovať použitie funkcie cin.get():

```
//retazce3.cc -- precitanie jedneho a viacej slov pomocou cin.get()
#include

using namespace std;

int main() {
        const int velkost=20;
        char meno[velkost];
        char dezert[velkost];

        cout << "Zadajte vase meno: \n";
        cin.get(meno,velkost).get(); //precita riadok a potom aj znak konca
riadku
        cout << "Zadajte vas oblubeny dezert: \n";
        cin.get(dezert,velkost).get();
        cout << "Mam vyborny "<< dezert;
        cout << ", ktory si zasluzi iba " << meno << ".\n";
        return 0;
}
```
Náš očakávaný výstup:

```
Zadajte vase meno:
Martin Bodocky
Zadajte vas oblubeny dezert:
Pernik domaci
Mam vyborny Pernik domaci, ktory si zasluzi iba Martin Bodocky.
```

Všimnite si, že C++ povoľuje viacnásobné volanie funkcií, ak keď majú rôzny zoznam argumentov. Prečo by sme mali používať cin.get() namiesto cin.getline? Funkcia cin.get() má viacej možností použitia. Predpokladajme, že čítame riadok do poľa. Ako zistíme či ho prečíta funkcia celý alebo skončí pri naplnenom poli? Pozrieme sa na znak ďalšieho prvku, a ak to je znak konca riadku, tak ho funkcia prečítala celý. Ak iné, tak nám chýbajú ešte ďalšie dáta. Ale túto problematiku budeme preberať v neskorších častiach kurzu. Stručne povedané funkcia cin.getline() je jednoduchšia na použitie ale cin.get() zjednodušuje ošetrenie chýb. Na načítanie riadku môžete použiť obidve z týchto funkcií, ale musíte myslieť na ich rozdielne chovanie.

Prázdne riadky a ďalšie problémy

Čo sa stane, keď funkcia cin.getline() alebo cin.get() prečíta prázdny riadok? Pôvodný postup bol taký, že nasledujúci príkaz vstupu začal tam, kde posledná funkcia vstupu skončila. Avšak v súčasnosti je po prečítaní prázdneho riadku funkciou cin.get() nastavený takzvaný chybový bit (*failbit*). Potom je ďalší vstup zablokovaný, a musíte ho odblokovať funkciou:

```
cin.clear();
```

Ďalšie problémy nastávajú, keď je reťazec dlhší ako blokovaný priestor. Ak je vstupný riadok dlhší, ako je stanovený počet obidve funkcie cin.getline() aj cin.get() nechávajú ostatné znaky vo fronte. A funkcia cin.getline() navyše nastaví chybový bit a vypne ďalší vstup.

Zmiešanie reťazového a číselného vstupu

Zmiešanie číselného a reťazového vstupu môže spôsobiť problémy. Urobíme si na to názorný program:

```
//retazce5.cc -- zmiesanie retazoveho a ciselneho vstupu
#include
using namespace std;

int main() {
        const int velkost=80;
        cout << "V akom roku si sa nastahoval?\n";
        int rok;
        cin >> rok;
        cout << "Aku mas adresu?\n";
        char adresa[velkost];
        cin.getline(adresa,velkost);
        cout << "Rok nastahovania: "<< rok<< "\n";
        cout << "Adresa:"<< adresa << "\n";
        cout << "finito! ;-)\n";
        return 0;
}
```

Výstup:

```
V akom roku si sa nastahoval?
1987
Aku mas adresu?
Rok nastahovania: 1987
Adresa:
finito! ;-)
```

Nikdy nedostaneme príležitosť zadať adresu. Tento problém je spôsobený tým, že keď cin prečíta rok zanechá vo fronte generovanú adresu ENTER čiže znak konca riadku. Potom cin.getline() prečíta znak konca riadku a priradí prázdny reťazec do poľa adresa. Oprava spočíva v prečítaní a zahodení znaku konca riadku pred čítaním adresy. Môžeme to urobiť niekoľkými spôsobmi, aj volaním cin.get() bez argumentu. Volanie môžeme uskutočniť oddelene:

```
cin >> rok;
cin.get();
```

Alebo spoločne:

```
(cin >> rok).get();
```

Náš výstup po úprave:

```
V akom roku si sa nastahoval?
1987
Aku mas adresu?
kolejni 2, praha 5
Rok nastahovania: 1987
Adresa:kolejni 2, praha 5
finito! ;-)
```