Znaky a reťazce, part 2: reťazce

Náhradná Prednáška 6, PROG-2, 2020
Pavol Zajac

1. Reťazce a reťazcové konštanty

Reťazce vo všeobecnosti sú postupnosti znakov. Viaceré programovacie jazyky majú špeciálne typy pre reťazce a operácie s nimi (napr. Python). V jazyku C je to trochu komplikovanejšie...

Reťazce v jazyku C sú uložené v pamäti ako postupnosť znakov (pole typu `char`). Platný reťazec by mal byť vždy ukončený špeciálnym znakom s ACII kódom nula, ktorý sa dá zapísať znakovou konštantou `'\0'`. V jazyku C s takto definovanými reťazcami pracujú rôzne knižničné funkcie, a prekladač podporuje špecifickú syntax na definovanie reťazcových konštánt: postupnosť znakov uzavretých v úvodzovkách `"`.

Ako to funguje:

1. Prekladač v kóde nájde reťazcovú konštantu: `"Priklad"`
2. V preloženom kóde do špeciálnej sekcie pamäte uloží príslušnú postupnosť znakov, a automaticky za ne doplní znak `'\0'`.

'P'	'r'	'i'	'k'	'l'	'a'	'd'	'\0'

3. Výsledkom výrazu so znakovou je potom adresa začiatku tejto pamäte, táto adresa odkazuje na nemeniteľnú postupnosť znakov, teda je typu `const char*`

Napr. keď používate `int printf(const char* format, ...)`, prvý argument s formátovacím reťazcom sa vyhodnotí ako adresa začiatku pamäte, kde bude tento formátovací reťazec uložený. Printf funkcia bude vidieť na vstupe túto adresu, uloženú v smerníku `format`.

2. Reťazcové premenné

Častou úlohou pri programovaní je meniť reťazce, vytvárať nejaké vlastné za behu programu, resp. načítať reťazce od používateľa. V jazyku C však na to nie je špecifický reťazcový typ, reťazce môžete vytvárať a meniť v rámci vopred alokovaného poľa znakov.

Príklad:

```
char pole[200];  //rezervovali sme si pamat na max. 200 znakov

//zapiseme do neho postupne malu abecedu
for (int i = 0; i < 26; i++)
    pole[i] = i + 'a';

//v tomto bode sa stale jedna o vseobecne pole znakov,
// kedze nie je ukoncene znakom '\0'
//nasledovnym prikazom z neho spravime platny retazec:

pole[26] = '\0';
```

Pozn.: jedno fyzicky alokované pole znakov môže obsahovať viacero logických reťazcov.

- na adrese `pole` sa nachádza reťazec `"abcdefghijklmnopqrstuvwxyz"`
- na adrese `pole+1` sa nachádza reťazec `"bcdefghijklmnopqrstuvwxyz"`
- ...
- na adrese `pole+25` sa nachádza reťazec `"z"`
- na adrese `pole+26` sa nachádza prázdny reťazec `""` (adresa odkazuje na ukončovací znak)
- na adrese `pole+27` sa nachádza nedefinovaná postupnosť znakov (keďže sme túto časť poľa nevyplnili, sú tam nejaké bajty v pamäti, ktorých obsah závisí od doterajšieho behu programu, prekladača a OS)

Keby sme napísali príkaz

```
pole[5] = '\0';
```

nahradili by sme znak `'f'` (prečo `'f'` a nie `'e'`?) ukončovacím znakom, ale zvyšok poľa by sa nezmenil. Potom by platilo:

- na adrese `pole` sa nachádza reťazec `"abcde"`
- na adrese `pole+1` sa nachádza reťazec `"bcde"`
- na adrese `pole+5` sa nachádza prázdny reťazec `""`
- na adrese `pole+6` sa nachádza prázdny reťazec `"ghijklmnopqrstuvwxyz"`

Keby sme chceli pracovať s reťazcom na adrese pole a reťazcom na adrese pole+6 ako so samostatnými reťazcami, môžeme si vytvoriť smerníky, kam zapíšeme príslušné adresy:

```
char *str1 = pole;    //odkazuje na retazec "abcde"
char *str2 = pole+6;  //odkaz na "ghijklmnopqrstuvwxyz"
```

Pozor! Premenné str1 a str2 sú smerníky, ktoré odkazujú na nejakú iným spôsobom alokovanú pamäť. Ak sa zmení obsah poľa, zmení sa aj obsah logických reťazcov, na ktoré str1 alebo str2 odkazujú. Ak sa uvoľní pamäť poľa, smerníky budú odkazovať na neplatnú pamäť.

Smerníky na char môžete použiť, aj keď chcete odkazovať viackrát na ten istý konštantný reťazec:

```
//odkazuje na prekladacom rezervovanu pamat
const char *autor = "Pavol";
```

Ak chcete vytvoriť pole znakov predvyplnené reťazcom, môžete použiť syntax reťazcových konštánt:

```
//vytvori sa pole 6 znakov, 5 na pismena, a 1 na ukoncenie retazca
char meno[] = "Pavol";
//vytvori sa pole 30 znakov, 5 sa naplni pismenami, a vsetky ostatne
nulami
char priezvisko[30] = "Zajac";
```

Na rozdiel od predošlého zápisu, obsah poľa meno sa dá meniť, znak po znaku. Napr.
`meno[3] = 'e';`

3. Dĺžka reťazca

Dĺžka reťazca je počet nenulových znakov od jeho začiatku po ukončovací znak, ten sa do dĺžky reťazca neráta.

Pozor: Kapacita poľa, resp. miesto, ktoré je potrebné v pamäti pre reťazec, je teda dĺžka reťazca + 1

Na určenie dĺžky reťazca nám postačuje adresa pamäťového miesta, kde reťazec začína. Mohli by sme napísať napr. funkciu:

```
int dlzka(const char *str)
{
      int len = 0;
      while (str[len] != '\0')
            len++;
      return len;
}
```

Pozn.: v príklade vstupom je pamäťová adresa. Ďalej s reťazcom pracujeme ako s poľom znakov, v ktorom hľadáme špeciálny ukočovací znak. Všimnite si, keďže vieme koniec reťazca identifikovať cez kontrolu znaku `'\0'`, tak nie je potrebné, aby funkcia dostala ako ďalší parameter dĺžku poľa.

Keďže úloha zistiť dĺžku reťazca je častá, štandardná knižnica poskytuje funkciu `strlen`, definovanú v hlavičkovom súbore `string.h`.

- úloha: pozrite si dokumentáciu k `string.h` a funkcie, ktoré poskytuje, máte ich spracované aj v úlohách na precvičenie reťazcov

Ako dostať reťazce od používateľa do programu, a ako ich vypísať na výstup?

4. Výpis reťazcov

Zobrazenie obsahu reťazca je pomerne jednoduché, `printf` funkcia priamo vypisuje konštantné reťazce, a podporuje výpis premenných ako reťazcov cez formátovací reťazec `%s`:

```
printf("Vitaj %s %s!\n", meno, priezvisko);
```

Pozn.: nepoužívajte priamy výpis premennej `printf(meno)`, ale použite `printf("%s", meno);` *Rozdiel je v tom, že vo formátovacom reťazci sa interpretujú znaky %, čiže ak premenná meno tento znak obsahuje, tak prvý výpis nie je definovaný (a môže spôsobiť bezpečnostné incidenty).*

Keď printf narazí na formátovací reťazec %s, pozerá sa na nasledujúci parameter ako na adresu. Začne z tejto adresy vypisovať znaky zaradom (vrátane špeciálnych, ako je napr. koniec riadku), až kým nenarazí na `'\0'` (tento znak sa nevypisuje). Ak napr. vytárate reťazec v poli a zabudnete ho ukončiť, tak sa printf pokúsi vypísať aj dáta mimo poľa (až kým nenarazí na nejaký nulový bajt).

Pozn.: Do printf môžete ako reťazec poslať ľubovoľnú adresu. Vyskúšajte, čo spraví
`printf("%s", main);`

5. Vstup z príkazového riadku

Keď spúšťate program v konzole (z príkazového riadku), operačný systém okrem spustenia programu do programu predpripraví argumenty. Príkazový riadok (vrátane mena programu) sa rozdelí na úseky oddelené medzerami, a tieto úseky sa použijú ako parametre funkcie main.

Funkcia main má preto ako vstup pole adries. Keďže sa jedná o pole adries, nie jeden reťazec, potrebuje aj jeho dĺžku /počet parametrov/.

Pozn.: Ak potrebujete, aby parameter obsahoval medzery, je potrebné ho dať do úvodzoviek.

Príklad:

```
// argc: pocet parametrov
// argv: je to pole, prvok pola ma typ char*, teda adresa nejakeho
znaku, resp. postupnosti znakov (retazca)
int main(int argc, char* argv[])
{
    for (int i = 0; i < argc; i++)
        printf("Arg%i: %s\n", i, argv[i]);
    return 0;
}
```

```
> program.exe -c 10 -f "hello world"

Arg0: program.exe
Arg1: -c
Arg2: 10
Arg3: -f
Arg4: hello world
```

Pozn.: V príklade sa môže zdať, že Arg2 je číslo. V skutočnosti sú to dva znaky, znak `'1'` a znak `'0'` (plus v pamäti je za nimi ukončovací znak `'\0'`, ktorý má iný ASCII kód ako znak *`'0'`*).

POZOR: na rozdiel od Pythonu, ak chcete skonvertovať reťazec na číslo, nestačí ho pretypovať cez `(int)`! Čo by sa stalo: pretypovali by ste adresu. Na preklad reťazca na číselnú hodnotu (int/double) je možné použiť hotové funkcie, ktoré sú deklarované v `stdlib.h` (napr. `atoi`)

- úloha: pozrite si dokumentáciu k `stdlib.h` a funkcie, ktoré poskytuje na konverzie reťazcov na čísla a opačne

Pozn.: Vo väčšine vývojových nástrojov sa dajú nastaviť parametre príkazového riadku, bez toho, aby bolo nutné skompilovaný program spúšťať ručne. Vyskúšajte si obe možnosti.

5. Vstup z konzoly

Na vstup reťazcov z konzoly je možné použiť viacero spôsobov. V každom prípade však potrebujeme mať **predpripravené pole znakov** postačujúcej veľkosti (neskôr sa dozvieme, ako sa dá toto pole rozšíriť, keby sme počas načítavania zistili, že kapacita nestačí): kapacita musí byť aspoň o 1 väčšia

ako dĺžka najdlhšieho potenciálneho vstupného reťazca.

Keďže sa jedná primárne o pole znakov, program môže doňho čítať znaky v cykle po nejakú ukončovaciu podmienku, a potom ručne ukončiť načítavanie:

```
int readline(char pole[], int maxlen)
{
     int i;

     //nacitavame maxlen znakov
     for (i = 0; i < maxlen; i++)
     {
          scanf("%c", &pole[i]);

          //max vsak po koniec riadku
          if (pole[i] == '\n')
              break;
     }

     //ked sme koniec riadku nedosiahli, je to chyba:
     if (pole[i] != '\n')
         return 0;

     //inak nahradime koniec riadku ukoncenim retazca,
     // a uspesne skoncime:
     pole[i] = '\0';
     return 1;
}

#define MAXLEN 1000
int main()
{
    char buffer[MAXLEN+1];

    if (!readline(buffer, MAXLEN))
         printf("Chyba vstupu, prilis vela znakov.\n");
    else
         printf("Nacital sa riadok dlzky %i:\n%s\n",
                                 strlen(buffer), buffer);

    return 0;
}
```

Kvôli tomu, že jazyk C nekontroluje indexy, je načítavanie údajov veľmi nebezpečné, obzvlášť pri reťazcoch. Ak by sme nekontrolovali vo funkcii readline dĺžku, ale načítavali až po koniec, tak by bolo možné vhodne skonštruovaným vstupom prepísať nielen buffer, ale aj iné úseky pamäte (a dostať tak napr. vírus do počítača...). Takto to robí napr. knižničná funkcia `gets`, ktorú určite nepoužívajte!

Knižničné funkcie na načítanie reťazca z konzoly, ktoré sú síce nebezpečné, ale dajú sa správne použiť

sú `scanf` a `fgets`.

Funkcia **`scanf`** podporuje formátovací reťazec %[WIDTH]s na načítanie jedného *slova* zo vstupu. Slovo je postupnosť alfanumerických znakov oddelených bielymi znakmi (white space = medzera, tab, koniec riadku). Biele znaky funkcia scanf do reťazca nenačíta. Pri čítaní sa vždy pridáva koncový znak, čiže za [WIDTH] sa má dosadiť číslo, ktoré je o 1 menšie ako (zostávajúca) kapacita poľa.

POZOR: vstupom do scanf je adresa, čiža ak použijeme názov poľa, alebo pointer, & sa nedáva.

```
char meno[31];
char priezvisko[31];

printf("Zadaj meno a priezvisko (max. 30 znakov kazde): ");
scanf("%30s %30s", meno, priezvisko);
```

Funkcia **`fgets`** je podobná príkladu z readline.

Deklarácia: `char* fgets(char *buffer, int n, FILE* stream);`

Vstupom **fgets** je buffer, kam sa má riadok načítať, a maximálny počet znakov (n je miesto vrátane ukončovacej nuly). Posledný parameter je vstupný súbor, ak chceme čítať z konzoly (klávesnice), tento parameter je treba nastaviť na globálnu premennú `stdin`. /Súbory budú neskôr/

Ak **fgets** je načíta maximálne n-1 znakov zo vstupu zaradom do pamäte s adresou buffer, vrátane znaku `'\n'` (koniec riadku), a potom pridá ukončovací znak. Ak je riadok dlhší ako predpoklad, dá sa to detekovať tak, že na konci chýba `'\n'`:

```
char buffer[81];
int len;

fgets(buffer, 81, stdin);
len = strlen(buffer);
if (len < 1 || buffer[len-1] != '\n')
    printf("Chyba vstupu\n");
```

Pozn.: `str[strlen(str)]` má vždy hodnotu `'\0'`. Posledný platný znak reťazca je `str[strlen(str)-1]`, okrem prípadu, že dĺžka reťazca je 0.

Funkcia fgets vracia adresu buffera, alebo keď nastala chyba vráti adresu 0 (adresa 0 je definovaná ak ako makro NULL). To sa využíva hlavne pri práci so súbormi, kedy vieme detekovať, že sme už prešli celý súbor.

5. Vybrané užitočné funkcie na prácu s reťazcami

Dôležité funkcie:

5.1. Kopírovanie obsahu:

Je potrebné si uvedomiť, že pri práci s reťazcami máte k dispozícii adresu, nie obsah, ten získate len po jednom znaku, buď cez operátor *, alebo cez indexy.

Keď napíšete niečo ako:

```
char str1[] = "Test";
char *str2 = str1;
```

tak vytvoríte dva odkazy na rovnakú pamäť. Zmena obsahu str1, napr.: `str1[0]='t';` by zmenila aj `str2` a opačne.

Naopak, keby ste napísali niečo ako:

```
char meno[31];
char prezyvka[31];
read(meno);         // nejaka funkcia, co ste si spravili...
prezyvka = meno;    // tu by ste chceli mat najprv kopiu mena
```

tak v jazyku C je to neplatné. Snažíte sa zapísať do premennej prezývka, ale to nie je smerník, ale pole, a teda sa nedá editovať kam odkazuje (daný príkaz znamená: zober adresu priradenú začiatku poľa meno, a zapíš ju do vhodného smerníka, ale prekladač tam smerník nemá...).

Skopírovanie obsahu je možné cez :

`strcpy, strncpy, memmove` – slúžia na presun údajov v pamäti, ak nechcete kopírovať údaje znak po znaku

`strcpy` : nebezpečné, lebo nekontroluje dĺžku, v tomto prípade ok:

```
strcpy(prezyvka, meno);
```

`strncpy` : dá sa obmedziť, koľko sa kopíruje, ale pozor na ukončovacie znaky... viď dokumentáciu

`memmove` : na rozdiel od predošlých funkcií, ktoré sa zastavia pri '\0' sa proste skopíruje, koľko bajtov špecifikujete (podobná funkcia je aj `memcpy`, ale tá nekontroluje, či sa náhodou pamäť neprekrýva)... viď dokumentáciu

5.2. Porovnávanie obsahu:

Podobne ako pri koírovaní, ak porovnáme dva reťazce nasledovne:

```
if (meno == prezyvka) ...;
```

tak vysledok bude v našom príklade vždy false (aj keby tam boli rovnaké texty), keďže sa porovnávajú adresy, nie obsahy polí. Na porovnanie celých reťazcov je možné použiť `strcmp`, na porovnanie častí reťazcov `strncmp`, a nájdenie podreťazca `strstr`.

Funkcie strcmp a strncmp vrátia celé číslo, ktoré určuje aj abecedné poradie reťazcov: pri prvej nezhode sa odpočítajú ASCII kódy znakov, a to je výstup. Čiže ak voláme

`strcmp("a", "b")`, výsledok bude < 0 (logicky `"a" < "b"`)

`strcmp("aaa", "aaa")`, výsledok bude 0 (logicky `"aaa" == "aaa"`)

`strcmp("aaa", "a")`, výsledok bude > 0 (logicky `"aaa" == "a"`)

Čiže ak chceme otestovať zhodu mena a prezývky, treba použiť:

```
if (strcmp(meno, prezyvka) == 0) ...;
```

Funkcia `strstr` sa používa, ak chcete zistiť, či sa jeden reťazec nachádza ako podreťazec iného:

```
strstr(prehladavam, hladam)
```

Ak sa v reťazci prehľadávanom nájde reťazec hľadaný, výsledkom je pointer do reťazca prehladavam (na mieste, kde zacina hladam), inak funkcia vráti pointer NULL. Ak chcete rýchlejšie hľadať výskyt 1 znaku, existuje aj funkcia `strchr`.

Pozn.: Všetky porovnávacie funkcie sú case sensitive, znak `'A'` *je iný ako* `'a'`*. Pozrite si príklad p06-04.c*

5.3. Ďalšie funkcie:

V jazyku C existuje množstvo štandardných funkcií, ktoré pracujú s reťazcami, nielen deklarovanými v string.h, ale aj stdlib.h, stdio.h, a mnohých ďalších. Je potrebné sa s nimi postupne zoznámiť na základe učebníc, dokumentácie, a pri ich používaní na riešenie úloh.

- úloha: minimálne je ešte potrebné poznať funkcie, ktoré sa používajú v časti venovanej reťazcom na stránke predmetu v Zbierke príkladov
  https://uim.fei.stuba.sk/b-prog2/b-prog2-zbierka-prikladov/

- posielam tiež v prílohe ukážky malých programov s komentovanou prácou s reťazcami

- na zadaní 3 je vhodné používať knižničné funkcie, aby ste nemuseli písať toľko vlastného kódu