Programátorská Encyklopedie

KORESPONDENČNÍHO SEMINÁŘE Z PROGRAMOVÁNÍ

Pokročilé parsování vstupu v jazyce C

V úvodním článku o parsování v jazyce C[1] jsme ukázali, jak efektně používat funkci `scanf()` k načítání čísel nebo stringů oddělených mezerami. V tomto pokročilejším článku ukážeme, jak si udělat rozdělení stringů podle oddělovačů sami a jak načítat vstup po jednotlivých znacích.

Načítání po znacích

Co dělat ve chvíli, kdy jsme postaveni před problém spočítat počet mezer na vstupu nebo nasekat vstup na slova, když nemáme omezení na jejich maximální délku? Tady nám už `scanf()` příliš nepomůže, budeme si to muset udělat sami.

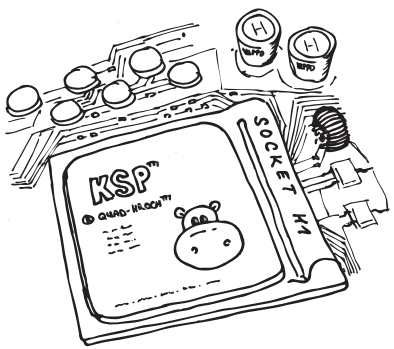
Jedno z možných řešení spočívá ve využití funkcí `fgetc()` nebo `fgets()` a procházení načteného stringu postupně po znacích. Obě funkce dostávají ukazatel na otevřený soubor, či jim lze předhodit třeba `stdin`.

První zmíněná funkce vždy přečte jeden znak a vrátí jeho hodnotu, nebo číslo -1, pokud již nelze načíst další znak. Spočítat počet mezer na vstupu by tak šlo třeba takto:

```
int pocet = 0;
int c;
while ((c = fgetc(vstup)) != -1) {
        if (c == ' ') pocet++;
}
```

Pozor je třeba dát, pokud kombinujeme třeba načítání čísel pomocí `scanf()` s načítáním jednotlivých znaků. Často se zapomíná na to, že znak nového řádku je také normálním znakem a při prostém zavolání `scanf("%d", &a)` nám na vstupu tento znak zůstane. A `fgetc()` (pro `fgets()` platí to samé) by ho pak přečetlo jako první znak a to by nám mohlo dělat neplechu.

Vyřešit se to dá přidáním mezery k formátovacímu stringu `scanf()`. Mezera ve formátovacím stringu totiž požere všechny bílé znaky (mezery, tabulátory a znaky nového řádku), na které na vstupu narazí, a zarazí se na prvním nebílém znaku: `scanf("%d ", &a)`.



Načítání stringů

Název `fgetc()` je odvozený od „file get char“, takže, jak už název druhé funkce `fgets()` napovídá, bude tato načítat stringy. Fungování je takové, že se do bufferu, který dostane jako první parametr, pokusí načíst buď celý řádek (včetně znaku konce řádku), nebo tolik znaků z něj, na kolik jí omezuje druhý parametr. Třetím parametrem je opět odkaz na otevřený soubor:

```
FILE *vstup = fopen("1.in", "r");
char buffer[1000];
fgets(buffer, 1000, vstup);
fclose(vstup);
```

Tím se nám do bufferu načetla posloupnost maximálně tisíce znaků ze vstupu. Je asi důležité v tuto chvíli pamatovat na to, že v jazyce C je textový řetězec vlastně polem znaků (pole typu `char`). To sebou nese dvě klíčové vlastnosti.

Za prvé, dá se přistupovat k jednotlivým znakům (vlastně přistupujeme na indexy v poli). Často si ale vyrobíme pole dlouhé třeba tisíc znaků a naše slovo jich bude mít méně. Abychom si nemuseli ještě někde bokem pamatovat, jak je naše slovo přesně dlouhé, je zvykem, že string v C vždy končí nulovým bajtem. Toho využívá například vestavěná funkce `strlen()`, která umí pole projít a spočítat, kolik znaků potkala před nulovým bajtem, neboli, jak je string dlouhý.

Druhou důležitou vlastností je to, že proměnná s polem je vlastně jen odkaz na nějaké místo v paměti, kde leží náš textový řetězec. Kdybychom si takovou proměnnou jen zkopírovali a změnili bychom její obsah, změnili bychom tím vlastně obsah původní proměnné.

Kopírování stringů je potřeba udělat vytvořením nového stringu a nakopírováním obsahu znak po znaku. Na to jsou již připravené vestavěné funkce `strncpy()` a `memcpy()`, které dostanou odkaz do paměti (klidně doprostřed pole znaků) od kterého nakopírují zadaný počet znaků (bajtů) do cílového umístění (v našem případě opět pole znaků).

První zmíněná funkce je speciálně určená ke kopírování stringů, o kterých nic nevíme (kontroluje průběžně, jestli nedošla na nulový bajt, a případně se na tomto místě zastaví), ale protože s sebou nese několik ukrytých nástrah a protože budeme o kopírovaných znacích mít více informací, použijeme raději `memcpy()`.

Jen při kopírování stringů obecně je nutné dát pozor na ukončení zkopírované stringu nulovým bajtem (doplnit ho na jeho konec či jinak zařídit, aby zde byl), pokud ho s ním nezkopírujeme už z původního stringu.

Na následujících řádcích ukážeme dvě techniky, jak nasekat dlouhý string na menší části podle mezer. Pro jednoduchost si dovolíme předpokládat, že vstupní řádek bude dlouhý maximálně 1000 znaků (pokud by byl příliš velký, dá se do bufferu načítat po částech, jen je nutné dávat pozor na slova přetékající hranici mezi částmi).

[1] `http://ksp.mff.cuni.cz/encyklopedie/parsovani-vstupu-c.html`

Rozdělení stringu zkopírováním

První technika používá zmíněné kopírování stringů. Pole `slova` je polem ukazatelů na pole znaků – vlastně tedy polem ukazatelů na slova, kde si každé slovo vytvoříme jen tak velké, jak potřebujeme. Pokud ještě úplně nerozumíte syntaxi ukazatelů, můžete kód brát jako magickou konstrukci která udělá to, co potřebujete:

```
char buffer[1000];
fgets(buffer, 1000, vstup);

char *slova[1000];
int slov = 0;

int N = strlen(buffer);
// Indexy: i - aktuální znak
//         zacatek - zapamatovaný začátek slova
int i, zacatek = 0;
for (i = 0; i <= N; i++) {
  if (buffer[i] == ' ' || buffer[i] == 0) {
    // Otestujeme délku posledního slova
    if(zacatek != i) {
      int d = i - zacatek;
      // Alokuji si prostor o jedna větší
      // kvůli koncovému nulovému bajtu
      slova[slov] = calloc(sizeof(char), d + 1);
      memcpy(slova[slov],      // Kam kopírovat
             &buffer[zacatek], // Odkud
             d);               // Kolik znaků
      slov++;
    }
    zacatek = i + 1; // Až za mezeru
  }
}
```

Tato technika je programátorům znalým jazyka C přímočará. Voláním `calloc()` si alokuji místo pro každé slovo a to si nakopíruji. Důvodem použití `calloc()` oproti běžnějšímu `malloc()` je ten, že `calloc()` mi navíc zaručuje, že bude přidělená paměť naplněná nulami. To může být při práci se stringy užitečné a navíc mi to tady umožňuje nestarat se o koncový nulový bajt – prostě si jen nechám vyrobit prostor o jedna větší, než délka slova.

Všimněte si navíc toho, že poslední slovo zpracujeme tak, že nulový bajt na konci bufferu bereme stejně, jako mezeru. Ušetříme si tím speciální ošetřování posledního slova až za cyklem.

Kdybych předem nevěděl, kolik je na vstupu slov, ještě bych si namísto pevně velkého pole `slova` mohl nejdříve spočítat slova prvním průchodem, pak si pole `slova` dynamicky vyrobit a teprve druhým průchodem provést skutečné nasekání. Ale tím jsme již nechtěli článek komplikovat a jistě si to zvládnete doplnit sami. :-)

Rozdělování stringu na místě

Druhá technika využívá triku s nulovými bajty. To, že v C končí stringy nulovým bajtem se totiž dá velmi pěkně využít. Opět budeme mít pole ukazatelů na jednotlivá slova, ale slova tentokrát zůstanou na místě, jen na správná místa bufferu (místo mezer) zapíšeme nulové bajty.

Když pak budeme se slovy pracovat, nebude nám to nijak vadit. Všechny funkce se zastaví na nulovém bajtu na konci slova a bude jim úplně jedno, že jen o bajt dál pokračuje další slovo.

```
char buffer[1000];
fgets(buffer, 1000, vstup);

char *slova[1000];
int slov = 0;

int N = strlen(buffer);
// Indexy: i - aktuální znak
//         zacatek - zapamatovaný začátek slova
int i, zacatek = 0;
for (i = 0; i <= N; i++) {
  // Otestujeme délku posledního slova
  if (buffer[i] == ' ' || buffer[i] == 0) {
    if (zacatek != i) {
      buffer[i] = 0;
      slova[slov++] = &buffer[zacatek];
    }
    zacatek = i + 1; // Až za mezeru
  }
}
```

Závěr

Oba výše uvedené kusy kódu jsou odolné vůči vícenásobným mezerám i proti tomu, když buffer začíná mezerou. Pokud máte zajištěno, že vstup bude vypadat pěkně, dají se kódy trochu zjednodušit. Obdobně se dají lehce modifikovat k tomu, aby sekaly vstup třeba podle středníků a aby se jinak chovaly ke dvěma středníkům za sebou (můžete tak lehce napsat parsovátko CSV souborů).

Zbývá zmínit ještě poslední užitečnou věc. Pokud se vám stane, že budete po rozsekání z předchozích odstavců potřebovat naparsovat nějaký string jako číslo, můžete k tomu využít opět funkci z rodiny `scanf()`. Přesněji funkci `sscanf()`, která nenačítá ze souboru ani ze standardního vstupu, ale čte ze stringu:

```
char muj_string[] = "25 12.6";
int a; float b;
sscanf(muj_string, "%d%f", &a, &b);
```

Tímto návodem jsme snad pokryli všechny věci potřebné k načítání stringů v jazyce C a snad jsme vám tím alespoň trochu pomohli.

Článek pro vás sepsal
Jirka Setnička

Programátorská Encyklopedie

KORESPONDENČNÍHO SEMINÁŘE Z PROGRAMOVÁNÍ

Pokročilé parsování vstupu v jazyce C

V úvodním článku o parsování v jazyce C[1] jsme ukázali, jak efektně používat funkci `scanf()` k načítání čísel nebo stringů oddělených mezerami. V tomto pokročilejším článku ukážeme, jak si udělat rozdělení stringů podle oddělovačů sami a jak načítat vstup po jednotlivých znacích.

Načítání po znacích

Co dělat ve chvíli, kdy jsme postaveni před problém spočítat počet mezer na vstupu nebo nasekat vstup na slova, když nemáme omezení na jejich maximální délku? Tady nám už `scanf()` příliš nepomůže, budeme si to muset udělat sami.

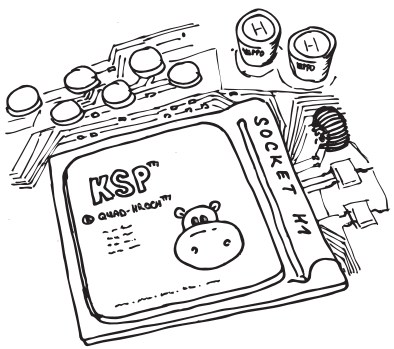
Jedno z možných řešení spočívá ve využití funkcí `fgetc()` nebo `fgets()` a procházení načteného stringu postupně po znacích. Obě funkce dostávají ukazatel na otevřený soubor, či jim lze předhodit třeba `stdin`.

První zmíněná funkce vždy přečte jeden znak a vrátí jeho hodnotu, nebo číslo -1, pokud již nelze načíst další znak. Spočítat počet mezer na vstupu by tak šlo třeba takto:

```
int pocet = 0;
int c;
while ((c = fgetc(vstup)) != -1) {
        if (c == ' ') pocet++;
}
```

Pozor je třeba dát, pokud kombinujeme třeba načítání čísel pomocí `scanf()` s načítáním jednotlivých znaků. Často se zapomíná na to, že znak nového řádku je také normálním znakem a při prostém zavolání `scanf("%d", &a)` nám na vstupu tento znak zůstane. A `fgetc()` (pro `fgets()` platí to samé) by ho pak přečetlo jako první znak a to by nám mohlo dělat neplechu.

Vyřešit se to dá přidáním mezery k formátovacímu stringu `scanf()`. Mezera ve formátovacím stringu totiž požere všechny bílé znaky (mezery, tabulátory a znaky nového řádku), na které na vstupu narazí, a zarazí se na prvním nebílém znaku: `scanf("%d ", &a)`.



Načítání stringů

Název `fgetc()` je odvozený od „file get char“, takže, jak už název druhé funkce `fgets()` napovídá, bude tato načítat stringy. Fungování je takové, že se do bufferu, který dostane jako první parametr, pokusí načíst buď celý řádek (včetně znaku konce řádku), nebo tolik znaků z něj, na kolik jí omezuje druhý parametr. Třetím parametrem je opět odkaz na otevřený soubor:

```
FILE *vstup = fopen("1.in", "r");
char buffer[1000];
fgets(buffer, 1000, vstup);
fclose(vstup);
```

Tím se nám do bufferu načetla posloupnost maximálně tisíce znaků ze vstupu. Je asi důležité v tuto chvíli pamatovat na to, že v jazyce C je textový řetězec vlastně polem znaků (pole typu `char`). To sebou nese dvě klíčové vlastnosti.

Za prvé, dá se přistupovat k jednotlivým znakům (vlastně přistupujeme na indexy v poli). Často si ale vyrobíme pole dlouhé třeba tisíc znaků a naše slovo jich bude mít méně. Abychom si nemuseli ještě někde bokem pamatovat, jak je naše slovo přesně dlouhé, je zvykem, že string v C vždy končí nulovým bajtem. Toho využívá například vestavěná funkce `strlen()`, která umí pole projít a spočítat, kolik znaků potkala před nulovým bajtem, neboli, jak je string dlouhý.

Druhou důležitou vlastností je to, že proměnná s polem je vlastně jen odkaz na nějaké místo v paměti, kde leží náš textový řetězec. Kdybychom si takovou proměnnou jen zkopírovali a změnili bychom její obsah, změnili bychom tím vlastně obsah původní proměnné.

Kopírování stringů je potřeba udělat vytvořením nového stringu a nakopírováním obsahu znak po znaku. Na to jsou již připravené vestavěné funkce `strncpy()` a `memcpy()`, které dostanou odkaz do paměti (klidně doprostřed pole znaků) od kterého nakopírují zadaný počet znaků (bajtů) do cílového umístění (v našem případě opět pole znaků).

První zmíněná funkce je speciálně určená ke kopírování stringů, o kterých nic nevíme (kontroluje průběžně, jestli nedošla na nulový bajt, a případně se na tomto místě zastaví), ale protože s sebou nese několik ukrytých nástrah a protože budeme o kopírovaných znacích mít více informací, použijeme raději `memcpy()`.

Jen při kopírování stringů obecně je nutné dát pozor na ukončení zkopírované stringu nulovým bajtem (doplnit ho na jeho konec či jinak zařídit, aby zde byl), pokud ho s ním nezkopírujeme už z původního stringu.

Na následujících řádcích ukážeme dvě techniky, jak nasekat dlouhý string na menší části podle mezer. Pro jednoduchost si dovolíme předpokládat, že vstupní řádek bude dlouhý maximálně 1000 znaků (pokud by byl příliš velký, dá se do bufferu načítat po částech, jen je nutné dávat pozor na slova přetékající hranici mezi částmi).

[1] `http://ksp.mff.cuni.cz/encyklopedie/parsovani-vstupu-c.html`

Rozdělení stringu zkopírováním

První technika používá zmíněné kopírování stringů. Pole `slova` je polem ukazatelů na pole znaků – vlastně tedy polem ukazatelů na slova, kde si každé slovo vytvoříme jen tak velké, jak potřebujeme. Pokud ještě úplně nerozumíte syntaxi ukazatelů, můžete kód brát jako magickou konstrukci která udělá to, co potřebujete:

```
char buffer[1000];
fgets(buffer, 1000, vstup);

char *slova[1000];
int slov = 0;

int N = strlen(buffer);
// Indexy: i - aktuální znak
//         zacatek - zapamatovaný začátek slova
int i, zacatek = 0;
for (i = 0; i <= N; i++) {
  if (buffer[i] == ' ' || buffer[i] == 0) {
    // Otestujeme délku posledního slova
    if(zacatek != i) {
      int d = i - zacatek;
      // Alokuji si prostor o jedna větší
      // kvůli koncovému nulovému bajtu
      slova[slov] = calloc(sizeof(char), d + 1);
      memcpy(slova[slov],      // Kam kopírovat
             &buffer[zacatek], // Odkud
             d);               // Kolik znaků
      slov++;
    }
    zacatek = i + 1; // Až za mezeru
  }
}
```

Tato technika je programátorům znalým jazyka C přímočará. Voláním `calloc()` si alokuji místo pro každé slovo a to si nakopíruji. Důvodem použití `calloc()` oproti běžnějšímu `malloc()` je ten, že `calloc()` mi navíc zaručuje, že bude přidělená paměť naplněná nulami. To může být při práci se stringy užitečné a navíc mi to tady umožňuje nestarat se o koncový nulový bajt – prostě si jen nechám vyrobit prostor o jedna větší, než délka slova.

Všimněte si navíc toho, že poslední slovo zpracujeme tak, že nulový bajt na konci bufferu bereme stejně, jako mezeru. Ušetříme si tím speciální ošetřování posledního slova až za cyklem.

Kdybych předem nevěděl, kolik je na vstupu slov, ještě bych si namísto pevně velkého pole `slova` mohl nejdříve spočítat slova prvním průchodem, pak si pole `slova` dynamicky vyrobit a teprve druhým průchodem provést skutečné nasekání. Ale tím jsme již nechtěli článek komplikovat a jistě si to zvládnete doplnit sami. :-)

Rozdělování stringu na místě

Druhá technika využívá triku s nulovými bajty. To, že v C končí stringy nulovým bajtem se totiž dá velmi pěkně využít. Opět budeme mít pole ukazatelů na jednotlivá slova, ale slova tentokrát zůstanou na místě, jen na správná místa bufferu (místo mezer) zapíšeme nulové bajty.

Když pak budeme se slovy pracovat, nebude nám to nijak vadit. Všechny funkce se zastaví na nulovém bajtu na konci slova a bude jim úplně jedno, že jen o bajt dál pokračuje další slovo.

```
char buffer[1000];
fgets(buffer, 1000, vstup);

char *slova[1000];
int slov = 0;

int N = strlen(buffer);
// Indexy: i - aktuální znak
//         zacatek - zapamatovaný začátek slova
int i, zacatek = 0;
for (i = 0; i <= N; i++) {
  // Otestujeme délku posledního slova
  if (buffer[i] == ' ' || buffer[i] == 0) {
    if (zacatek != i) {
      buffer[i] = 0;
      slova[slov++] = &buffer[zacatek];
    }
    zacatek = i + 1; // Až za mezeru
  }
}
```

Závěr

Oba výše uvedené kusy kódu jsou odolné vůči vícenásobným mezerám i proti tomu, když buffer začíná mezerou. Pokud máte zajištěno, že vstup bude vypadat pěkně, dají se kódy trochu zjednodušit. Obdobně se dají lehce modifikovat k tomu, aby sekaly vstup třeba podle středníků a aby se jinak chovaly ke dvěma středníkům za sebou (můžete tak lehce napsat parsovátko CSV souborů).

Zbývá zmínit ještě poslední užitečnou věc. Pokud se vám stane, že budete po rozsekání z předchozích odstavců potřebovat naparsovat nějaký string jako číslo, můžete k tomu využít opět funkci z rodiny `scanf()`. Přesněji funkci `sscanf()`, která nenačítá ze souboru ani ze standardního vstupu, ale čte ze stringu:

```
char muj_string[] = "25 12.6";
int a; float b;
sscanf(muj_string, "%d%f", &a, &b);
```

Tímto návodem jsme snad pokryli všechny věci potřebné k načítání stringů v jazyce C a snad jsme vám tím alespoň trochu pomohli.

Článek pro vás sepsal
Jirka Setnička