Programátorská Encyklopedie

KORESPONDENČNÍHO SEMINÁŘE Z PROGRAMOVÁNÍ

Základní parsování vstupu v jazyce C

Začínající programátoři se velmi často potýkají s tím, jak by měli do svého programu dostat vstup a jak ho zpracovat. V tomto článku a v sesterském článku o Pythonu 3[1] se vám s tím pokusíme pomoci. Úvodní kapitolu mají oba články stejnou, ale poté se již každý zabývá svým jazykem.

Klávesnice versus soubory

Nejjednodušší je ve většině jazyků načítat vstup od uživatele. To je dobré pro rychlé testování nebo pro programy přijímající jenom jednu vstupní hodnotu. Ale co když program přijímá na vstupu více hodnot?

Jedna možnost je změnit jeho vnitřnosti tak, aby místo toho načítal vstup ze souboru. Ale nejrychlejší většinou bývá zachovat program tak, jak je, a jen mu místo vstupu z klávesnice předhodit soubor.

Programu totiž většinou neví o klávesnici nic, má jen věc, které se říká *standardní vstup*. Ten je většinou představovaný klávesnicí (obdobně je pak *standardní výstup* většinou představovaný vypisováním na terminál), ale dá se jednoduše přesměrovat.

Jak na to? Představme si, že máme nějaký svůj program (spustitelný zkompilovaný program v C nebo skript v Pythonu) a soubor `1.in`, který mu chceme předhodit na vstupu. V příkazové řádce všech hlavních operačních systémů (Linux a obecně všechny UNIXy, Windows) to uděláme pomocí operátoru přesměrování ve tvaru menšítka (`<`):

```
./program < 1.in           -- UNIXový svět
program.exe < 1.in         -- Windows
python3 program.py < 1.in  -- oboje
```

(Pozor: V UNIXovém světě se při spuštění programu z aktuálního adresáře musí explicitně uvést adresář tečka jako odkaz na aktuální umístění.)

Stejným způsobem pak lze přesměrovat i výstup, jen s použitím šipky na druhou stranu (většítka). Pokdu tedy budete řešit třeba opendatové úlohy v KSP a budete mít program v Pythonu 3, může celý příkaz vypadat jako:

```
python3 program.py < 1.in > 1.out
```

Zpracování standardního vstupu

Hlavním pomocníkem programátora v C by měla být skoro všemocná funkce `scanf()`. Ta přijímá více parametrů: první z nich je formátovací string obsahující definice, které *tokeny* očekáváme na vstupu. Jako token můžeme chápat třeba celé číslo, desetinné číslo nebo string. Zbylé parametry jsou pak odkazy na místa, kam se mají tokeny uložit.

Formátovací string pro funkci `scanf()` je stejný, jako pro `printf()`, a mezi hlavní položky patří níže uvedené, další si snadno dohledáte v manuálu.

- `%d` – zástupný symbol pro celé číslo
- `%f` – zástupný symbol pro float
- `%lf` – zástupný symbol pro double
- `%s` – zástupný symbol pro string

Pokud tedy chci načíst ze vstupu třeba čísla oddělená mezerou, mohu použít následující konstrukci (všimněte si, že proměnné musím předávat odkazem, aby do nich mohla funkce `scanf()` uložit načtené hodnoty):

```
int a, b;
scanf("%d %d", &a, &b);
```

Pokud je ve formátovacím stringu očekáváno číslo nebo string, `scanf()` automaticky ořeže všechny *bílé znaky* (neboli mezerník, tabulátor a znak nového řádku) a pokusí se naparsovat první nalezené slovo jako číslo, respektive string. Obě ukázky níže je tedy ekvivalentní s ukázkou výše:

```
int a, b;
scanf("%d", &a);
scanf("%d", &b);
```

```
int a, b;
scanf("%d%d", &a, &b);
```

Při načítání stringu je volání odlišné v jedné věci. String v jazyce C je vlastně jen polem charů (jednotlivých znaků) a jako každé pole v C je vlastně již odkazem samo o sobě. Proto se před něj nemusí psát znak reference:

```
char jmeno[20];
scanf("%s", jmeno);
```

Jediná věc, na kterou je při čtení stringů třeba dát pozor je, aby se vešly do místa, které jsme pro ně připravili. Jak to dělat, pokud o vstupu nemáme slíbeno nic, nebo chceme vstup zpracovávat jinak je zmíněno v pokročilejším článku, na který odkazujeme na konci.

Načítání v cyklu

Díky vlastnostem zmíněným výše je možné `scanf()` efektivně používat pro načítání hodnot v cyklu bez toho, aniž bychom se museli starat o nějaké bílé znaky. Dá se napsat i cyklus, který bude načítat vstup, dokud na něm něco bude (respektive dokud se na vstupu neobjeví znak EOF nebo End-of-file, ten se dá napsat ručně třeba stiskem Ctrl+D v UNIXu nebo Ctrl+Z ve Windows).

Funkce `scanf()` totiž jako svoji návratovou hodnotu vrací počet tokenů, které se jí povedlo naparsovat. Pokud se při čtení dojde až na konec souboru a nepovede se načíst žádný token, vrátí hodnotu EOF reprezentovanou číslem -1. Nejčistší řešení je tedy kontrolovat, že jsme načetli přesně požadovaný počet tokenů (což může být důležité třeba při načítání dvojic čísel, kdy asi nechceme pokračovat dál jen s jedním z nich):

```
int a, b;
while(scanf("%d %d", &a, &b) == 2)
    // dělej něco
```

Bohužel v C není žádná jednoduchá vestavěná konstrukce, jakou umí Python s funkcí `.split()`, ale díky mocnosti funkce `scanf()` se bez toho dá většinou obejít. Výhodou navíc je, že takto lze zpracovávat libovolně velký vstup, protože v paměti je vždy jen jeho malá část. Jazyk C sice obsahuje podobně se chovající funkci `strtok()`, ale před

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

jejím používáním doporučujeme skutečně důkladně nastudovat její dokumentaci.

Čtení ze souborů

Práce se souborem se v C skoro neliší od zpracování standardního vstupu. Jen místo `scanf()` použijeme funkci `fscanf()` (podobně existuje třeba `fprintf()`), které jako první parametr předáme odkaz na otevřený soubor.

Jak si takový pořídit? Velmi jednoduše, například celé načtení jednoho čísla ze souboru může vypadat následovně:

```
FILE *vstup = fopen("1.in", "r");
int a;
fscanf(vstup, "%d", &a);
fclose(vstup);
```

Volání `fopen()` předáváme cestu k souboru a mód otevření. Parametr `r` otevírá jen pro čtení (parametr `w` pak pro zápis a parametr `a` pro přidávání na konec). Na konci práce se souborem je vhodné zavolat `fclose()` a soubor zavřít.

Jako perličku na konec si dovolíme dodat, že pokud souborovým funkcím předáme jako odkaz na soubor hodnotu `stdin` nebo `stdout`, pracují pak se standardním vstupem a výstupem (a nyní už také možná tušíte, proč je v C práce s nimi a se soubory tak podobná).

Závěr

V tomto článku jsme uvedlo základní parsování stringu v jazyce C, které vám pro většinu použití snad bude stačit. Pokud ale budete potřebovat načítat stringy znak po znaku, či rozdělovat vstupní string, u kterého nebudete vědět maximální délku jednotlivých slov, je nutné použít pokročilejší techniky ukázané v našem článku o pokročilejším parsování v jazyce C.[2]

Článek pro vás sepsal
Jirka Setnička

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