Primo Progetto del Laboratorio di Programmazione
Calcolo di espressioni aritmetiche

Stefano Guerrini

A.A. 2001/02

L'obiettivo del progetto è leggere, memorizzare in un albero e determinare il valore di espressioni aritmetiche intere.

Per semplificare la lettura delle espressioni, assumiamo che nella scrittura delle espressioni non si possano utilizzare le usuali regole di semplificazione delle parentesi, ma che tutte le parentesi vadano scritte. Pertanto, ogni espressione dovrà avere una delle seguenti forme

```
costante
variabile
( espressione operatore espressione )
```

dove

1. `costante` è una costante numerica intera non negativa (ad esempio, $0, 2, 5, 18, 1256$). Per semplificare la lettura delle espressioni, omettiamo le costanti negative, altrimenti il simbolo `-` dovrebbe avere due interpretazioni, una come operatore unario che cambia il segno di un numero ed una come operatore che indica la differenza di due numeri.

2. `variabile` è una sequenza di caratteri alfanumerici che comincia con un carattere alfabetico (si può assumere che i nomi delle variabili abbiano una lunghezza inferiore a 256).

3. Il terzo caso corrisponde ad una operazione binaria, dove `operatore` è uno tra `+`, `-`, `*` o `/`, mentre le espressioni a sinistra e destra dell'operatore sono gli operandi dell'operazione. Si osservi che l'impossibilità di omettere parentesi impone che le espressioni ottenute mediante una operazione devono essere sempre racchiuse tra una coppia di parentesi. Ad esempio, l'espressione che con le usuali regole per la semplificazione delle parentesi scriveremmo $a + b * c$, deve essere scritta $(a + (b * c))$.

Si osservi che durante la lettura delle espressioni è possibile capire qual è il caso da analizzare semplicemente vedendo il primo carattere dell'espressione da leggere: se è una cifra decimale, allora si tratta di una costante, se è una lettera allora si tratta di una variabile, se è una parentesi tonda aperta, allora si tratta di una espressione composta.

Le espressioni da leggere e valutare verranno prese da un file contenente una sequenza di assegnazioni della forma

```
variabile = espressione;
variabile = espressione;
...
variabile = espressione.
```

(si osservi che la sequenza è terminata da `.` e che le assegnazioni della sequenza sono separate da `;`).

Ogni assegnazione associa il valore della espressione alla destra del simbolo `=` alla variabile alla sinistra dell'`=`. Le assegnazioni nel file di input dovranno essere lette e valutate in ordine. Al momento della valutazione di un'assegnazione, alle variabili che appaiono nell'espressione deve essere già stato assegnato un valore da una precedente assegnazione, altrimenti si ha un errore.

Per mantenere i valori delle variabili si dovrà utilizzare una tavola hash.

Cosa si deve realizzare

1. Implementare la tavola hash per la memorizzazione delle variabili e dei loro valori. A tale fine si dovrà realizzare un modulo `hash.c` con header `hash.h` che contiene tutte le funzioni relative all'implementazione della tavola hash. In particolare, si vuole che questo modulo permetta la gestione di più tavole hash simultaneamente (anche se nel progetto ne serve solo una) e di poter fissare la dimensione della tavola da usare. Per questo motivo, la tavola hash, oltre a contenere il vettore con i puntatori alle liste dovrà mantenere anche la dimensione di tale vettore. Si richiede così di scrivere la funzione che crea ed inizializza una tavola; la funzione che inserisce una chiave ed il valore associato in una data tavola se la chiave non era presente nella tavola, o che aggiorna il valore associato alla chiave se questa era già presente nella tavola; la funzione che ricerca una chiave nella tavola e, nel caso di elemento presente, ne restutuisce il valore.

   Per la funzione di hash si consiglia la seguente semplice funzione

   $$\sum_{i=1}^{k} c_i \cdot 256^i \mod N$$

   dove $c_0 c_1 \ldots c_k$ sono i codici ASCII dei primi k caratteri della chiave ed N è la dimensione della tavola. (Attenzione ai problemi di overflow nel calcolo della funzione. A tale proposito si ricorda che $(a + b) \mod N = ((a \mod N) + (b \mod N)) \mod N$).

   Per la tavola hash del progetto si consiglia di assegnare un numero primo ad N.

2. Scrivere la funzione che traduce una espressione nell'albero corrispondente e la funzione che calcola il valore dell'espressione memorizzata nell'albero.

3. Scrivere un main che legge dalla linea di comando il nome del file in cui si trovano le assegnazioni o, nel caso di linea di comando vuota, legge le assegnazioni dallo standard input (vedi appendice A) e stampa la sequenza variabile/valore corrispondente alla sequenza di assegnazioni lette. Ad esempio, se il file di input è

   ```
   alfa = 3;
   beta = 10;
   x    = (alfa * beta);
   y    = ((alfa * x) + beta)).
   ```

   allora il programma deve stampare

   ```
   alfa 3
   beta 10
   x 30
   y 100
   ```

A Come acquisire il nome del file di input

Qui di seguito riportiamo una possibile implementazione della parte di main che deve acquisire il nome del file di input dalla linea di comando. Questa soluzione, nel caso non venga fornito alcun nome di file associa la variabile `fin` allo standard input, altrimenti prova ad aprire il file indicato sulla linea di comando associandolo alla varibile `fin`. Quindi, in ogni caso, dopo l'esecuzione del codice riportato qui sotto, il programma legge i suoi dati da `fin`.

```
int main(int argn, char *argv[]) {
    FILE *fin;
    ...
    ...

    /* elimina dalla lista dei parametri il nome del programma */
```

```
    argn--, argv++;

    if (argn == 0) /* non e' stato fornito alcun nome di file */
        fin = stdin;
    else { /* apri il file fornito nella linea di comando */
        if ((fin = fopen(*argv,"r")) == NULL)
            printf("Impossibile aprire il file \"%s\"\n", *argv);
    }

   ...
   ...

}
```