

Laboratorio di Algoritmi e Strutture Dati

Esercitazioni del 9 Novembre 2006

Esercizio 2: input, minicalc

Si tratta di un esercizio sull'input/output orientato ai caratteri, dunque sull'uso di `getchar` e `putchar`.

Nozioni utili

La libreria standard fornisce le seguenti funzioni e/o macro:

- `int isspace(int c)`
Restituisce un (non meglio specificato) valore > 0 se `c` è un carattere di *spaziatura*. Restituisce 0 altrimenti.
Richiede l'inclusione del file di intestazione `ctype.h`.
Nota: I caratteri di spaziatura sono: ' ', '\t', '\n', '\r', '\f', '\v', vale a dire, rispettivamente: spazio, tabulazione, newline, carriage return, form feed, tabulazione verticale.
- `int isalnum(int c)` restituisce un (non meglio specificato) valore > 0 se `c` è un carattere *alfanumerico*. Restituisce 0 altrimenti.
Richiede l'inclusione del file di intestazione `ctype.h`.
- `int isdigit(int c)`
Restituisce un (non meglio specificato) valore > 0 se `c` è una cifra. Restituisce 0 altrimenti.
Richiede l'inclusione del file di intestazione `ctype.h`.
- `int ungetc(int c, FILE *fp)`
Immette `c` nel buffer connesso al `FILE *fp`, in modo tale che la prossima lettura da questo file restituisca `c`.
Restituisce `c` oppure `EOF` se l'operazione non ha successo.
Richiede l'inclusione del file di intestazione `stdio.h`.
`ungetc` è utile quando ci si accorge di aver letto tutta la parte di input che ci si è prefissi di leggere solo dopo aver letto un ulteriore carattere. Utilizzando `ungetc` possiamo *rimettere* questo carattere nel buffer di input.
- `int strcmp(const char *s, const char *t)`
Restituisce un valore < 0 se `s` precede `t` nell'ordine lessicografico (quello del dizionario), restituisce 0 se `s` è uguale a `t`, restituisce un valore > 0 se `s` segue `t` nell'ordine lessicografico.
Richiede l'inclusione del file di intestazione `string.h`.

Prima parte: input

Implementare le seguenti funzioni, senza preoccuparsi di gestire la fine del file.

- `void saltaspazi(void):`
salta tutti i caratteri di *spaziatura* consecutivi presenti nell'attuale buffer di input.
Il primo carattere letto dopo una chiamata a `saltaspazi` non deve essere un carattere di spaziatura.
- `char leggispazio(void):`
Legge un carattere e restituisce 1 se il carattere letto è di spaziatura, altrimenti restituisce 0.

- **char leggi parola(char *word):**
Restituisce 0 se il prossimo carattere nel buffer di input non è *alfanumerico*. Altrimenti, legge la sequenza di caratteri *alfanumerici* consecutivi presenti nell'attuale buffer di input e la memorizza nella stringa **word** (non dimenticate di gestire il carattere di terminazione!). Alla fine restituisce 1. Il primo carattere letto dopo una chiamata a **leggi parola** non deve essere un carattere alfanumerico.
Nota: ovviamente **word** definita nell'ambiente chiamante è un array di **char** di una certa lunghezza **SIZEWORD** (diciamo **#define SIZEWORD 80**). Per questo esercizio assumiamo che ogni parola letta in input non sia più lunga di **SIZEWORD**.
- **int legginumero_nonnegativo(void):**
Restituisce -1 se il prossimo carattere nel buffer di input non è una cifra.
Altrimenti legge la sequenza di cifre consecutive presenti nell'attuale buffer di input, calcola il valore intero corrispondente, e lo restituisce.
Il primo carattere letto dopo una chiamata a **legginumero_nonnegativo** non deve essere una cifra.
- **char leggi_si_no(char *msg):**
Stampa la domanda **msg (s/n)**.
Se l'utente immette 's' restituisce 1.
Se l'utente immette 'n' restituisce 0.
Altrimenti, stampa il messaggio: **Input non corretto. Ripetere prego.**, ripete la domanda **msg (s/n)** e attende la risposta dell'utente.
Il ciclo viene ripetuto fino a quando l'utente non immette 's' oppure 'n'.
NOTA: alla fine di ogni ciclo potrebbe essere opportuno svuotare il buffer di input per ripartire con un input pulito, a questo scopo si consiglia di definire e usare la macro
#define svuotainput() while(getchar() != '\n')

Testare tali funzioni con il seguente main:

```
int main(void)
{
    int x;
    char w[SIZEWORD + 1];

    if(!leggi_si_no("Procediamo ?"))
        return -1;
    saltaspaзи();
    x = legginumero_nonnegativo();
    if(!leggispazio()) {
        printf("x = %d.\n",x);
        return 0;
    }
    saltaspaзи();
    if(leggi parola(w))
        printf("x = %d, w = %s.\n",x,w);
    else
        printf("x = %d.\n",x);
    return 0;
}
```

e con input adeguati (fa parte dell'esercizio trovare input adeguati a testare il comportamento del programma nei vari casi).

Seconda parte: minicalc

Utilizzare le funzioni implementate nella prima parte per implementare la seguente *minicalcolatrice*.

1. Viene stampato il messaggio:
`Immetti l'espressione:.`
2. Viene letta una riga di input della forma x *operazione* y , dove x e y sono numeri interi non negativi (con un numero arbitrario di cifre) e *operazione* o è la stringa **piu** o è la stringa **per**.
3. Se l'input non è conforme alle specifiche stampa il messaggio
`Input non corretto. Ripetere prego.`
e va al punto 6 (E' opportuno svuotare il buffer di input per ripartire con un input pulito).
4. Se *operazione* è **piu** stampa il messaggio:
`Risultato: z`
dove $z = x + y$.
5. Se *operazione* è **per** stampa il messaggio:
`Risultato: z`
dove $z = x * y$.
6. Stampa il messaggio
`Vuoi immettere un'altra espressione? (s/n)` e attende la risposta **s/n** dell'utente.
7. Se la risposta è **'s'** itera dal punto 1.

Ad esempio, se l'utente immette la riga di input:

`12 per 12`

il programma deve stampare:

`Risultato: 144`

`Vuoi immettere un'altra espressione? (s/n)`