Lezione 34 C e assembly 12 + I/O + preprocessore

martedì 5 marzo 2024 11:07

Andiamo ora a vedere come fare i/o con/da dispositivi . In C si usa un costrutto standard , ovvero la funzione **scanf()**: ovvero una funzione variadica che accetta un numero di parametri , ma attenzione al numero degli specificatori. Quindi questa funzione permette di leggere dei caratteri dal canale di i/o : li converte secondo lo specificatore passato ed infine memorizza i valori dentro questi specificatori. **Attenzione però al comportamento indefinito !!**. Per comportamento indefinito si intende se il numero di argomenti è maggiore di quello degli specificatori, oppure il contrario . **Nota: per il printf/scanf la stringa termina con '\0' , mentre per stringa normale (%s) termina con '\0'.** Vediamo degli esempi : Ricordiamo gli header e che è stato realizzato dentro uno switch:

```c
case 1:
{
        //se con input numerico funziona
        //sia positico che negativi
        //se con lettere return 0:
        // non processa input
        // -> stack vuoto
        //undefined behaviour
        int a;
        printf("enter a number: ");
        scanf("%d", &a);
        printf("You entered %d.\n", a);
        break;
}
```

```c
case 2:
{
        //con numeri funziona
        //con caratteri entra
        //in ciclo infinito!!
        int a;
        printf("enter a number: ");
        //si usa solo per scan formattate (da file)
        while (scanf("%d", &a) != 1) {
        // input was not a number, ask again:
                printf("enter a number: ");
        }
        printf("You entered %d.\n", a);
        break;
}
```

```c
case 3:
{
        //scrive dati su stdin->
        //potrebbe portare ad incosistenza
        int a;
        printf("enter a number: ");
        while (scanf("%d", &a) != 1) {
        // input was not a number, ask again:
        fflush(stdin); // <- never ever do that!
        printf("enter a number: ");
        }
        printf("You entered %d.\n", a);
        break;
}
```

```c
case 4:
{
        //se nome < 12 char ok
        //se nome con spazio solo il primo
        //se nome con separatore !=" " tutto
        char name[12];
        printf("What's your name? ");
        scanf("%s", name);
        printf("Hello %s!\n", name);
        break;
}
```

```c
case 5:
{
        //cosi funziona anche se spazi come separatore
        char name[40];
        printf("What's your name? ");
        scanf(" %39[^\n]", name);
        // note the space here, matching any whitespace
        printf("Hello %s!\n", name);
        break;
}
```

```c
case 6:
{
        //soffre l'enter
        char name[40];
        printf("What's your name? ");
        if (fgets(name, 40, stdin)) {
        printf("Hello %s!\n", name);
        }
        break;
}
```

```c
case 7:
{
        //trova indice che matcha con '\n'
        char name[40];
        printf("What's your name? ");
        if (fgets(name, 40, stdin)) {
                name[strcspn(name, "\n")] = 0;
                printf("Hello %s!\n", name);
        }
        break;
}
```

```c
case 8:
{
        //qui se input è carattere/stringa entra
        // in ciclo finche non trova un numero
        //funziona anche se '\n' oppure '\0'
        int a;
        char buf[1024]; // use 1KiB just to be sure
        do {
        printf("enter a number: ");
        if (!fgets(buf, 1024, stdin)) {
        // reading input failed, give up:
                return 1;
        }
        // have some input, convert it to integer:
        a = atoi(buf);
        } while (a == 0); // repeat until we got a valid number

        printf("You entered %d.\n", a);
        break;
}
```

```c
case 9:
{
        //questa versione migliorata bypassa il bug del '\n' e del '\0'
        long a;
        char buf[1024]; // use 1KiB just to be sure
        int success; // flag for successful conversion
        do {
        printf("enter a number: ");
        if (!fgets(buf, 1024, stdin)) {
        // reading input failed:
        return 1;
        }
        // have some input, convert it to integer:
        char *endptr;
        errno = 0; // reset error number
        a = strtol(buf, &endptr, 10);
        if (errno == ERANGE) {
        printf("Sorry, this number is too small or too large.\n");
        success = 0;
        } else if (endptr == buf) {
        // no character was read
        success = 0;
        } else if (*endptr && *endptr != '\n') {
        // *endptr is neither end of string nor newline
        // so we didn't convert the *whole* input
        success = 0;
        } else {
        success = 1;
        }
        } while (!success); // repeat until we got a valid number
        printf("You entered %ld.\n", a);
        break;
}
```

```c
default:
{
        printf("\n\tLeggi cosa vuoi fare ?? \n");
        printf("\n");
        printf("\t1) leggi un numero \n");
        printf("\t2) leggi un numero con ciclo\n");
        printf("\t3) leggi un numero e mandalo su stdin \n");
        printf("\t4) leggi una stringa \n");
        printf("\t5) leggi una stringa formattata \n");
        printf("\t6) leggi qualcosa \n");
        printf("\t7) leggi ed identifica terminatori \n");
        printf("\t8) leggi un numero senza gestione \n");
        printf("\t9) leggi un numero con gestione \n");
        printf("\n\n");
}
```

Ricordiamo che errno (error number) mantiene un riferimento ad ultimo errore verificato, mentre strol (str to long) trasfroma da byte alfanumerici a numerici. In generale come parametri dello strtol sono : un buffer dove scrivere, l'indirizzo della stringa che deve convertire, mentre l'ultimo rappresenta la base nella quale fare la conversione. In alternativa allo scanf vi sarebbe altra funzione : **la gets(), ma non si usa in quanto è piena di bug**. Andiamo ora a vedere invece il **preprocessore** C: ovvero un preprocessore di macro per i linguaggi di programmazione : più in generale si intendono le direttive di compilazione , le quali supportano anche la gestione del codice multi piattaforma. In dettaglio :

1. Inclusione di file
    a. Permette di includere file esterni a quel sorgente : se il file è tra "<" e">" lo cerca nei percorsi standard del compilatore, altrimenti se incluso tra " " lo cerca nella stessa cartella del sorgente corrente
2. Compilazione condizionale
    a. Servono a dire al compilatore di compilare secondo una specifica piuttosto che un'altra :

```
#ifdef __unix__
# include <unistd.h>
#elif defined _WIN32
# include <windows.h>
#endif
```

    ii. Permette di includere header unistd.h se SO è linux, altrimentiwindows.h se è windows
3. Guardie di inclusione
    a. Consentono di scegliere se e quando includere i file, quindi di definire simboli e tipi
    b.

```
#ifndef _FILENAME_H
#define _FILENAME_H
...
#endif
```

```
#pragma once
```

    c. Con la #pragma once si intende che il file sarà **compilato una ed una sola volta**.
4. Macro
    a. Servono a definire delle "costanti" : o variabili oppure funzioni. Una volta definite ogni riferimento a quella variabile eseguirà quel codice , oppure avrà quel valore.

b.

```c
#include<stdlib.h>
#include<stdio.h>

#define MAX(a,b) ((a) > (b) ? (a) : (b))
#define MIN(a,b) ((a) < (b) ? (a) : (b))

//per escludere una macro si usa #undef<macro>

int main(void)
{
        int x=20;
        int y=10;
        int max=MAX(x++,y);
        int min=MIN(x,y++);
        printf( "max tra x++: %d e y:%d  %d\n",x,y,max);
        printf( "min tra x:%d e y++:%d  %d\n",x,y,min);
        return 0;
}
```

c. Con output il seguente:

d.

```
max tra x++: 22 e y:12  21
min tra x:22 e y++:12  11
```