Fondamenti di Informatica e Laboratorio T-AB
Ingegneria Elettronica e Telecomunicazioni e
Ingegneria dell'Automazione
a.a. 2009/2010

Errori comuni

Versione Corretta

```
#include <stdio.h>

int main(void) {
    int limit, num;
    int i, j;
    int part, tot;
    printf("Quanti numeri vuoi inserire? ");
    scanf("%d", &limit);

    tot = 0;
    for (i=0; i<limit; i++) {
          printf("Inserisci il numero %d:", i+1);
          scanf("%d", &num);
          part = 1;
          for (j=num; j>0; j--)
                  part=part*j;
          tot = tot + part;
    }
    printf("Totale: %d\n", tot);
    return (0);
}
```

Versione Errata


```
#include <studio.h>

int main(void) {
    int limit, num;
    int i, j;
    int part, tot;
    printf("Quanti numeri vuoi inserire? ");
    scanf("%d", &limit);

    tot =
            part = 1;
            for (j=num; j>0; j--)
                    part=part*j;
            tot = tot + part;
    }
    printf("Totale: %d\n", tot);
    return (0);
}
```

```
cpp: c:\Users\alessio\desktop\main.c:1 Could not find include file <studio.h>
c:\Users\alessio\desktop\main.c: 6  missing prototype for printf
c:\Users\alessio\desktop\main.c: 7  missing prototype for scanf
Compilation + link time:0.2 sec, Return code: 0
```

Versione Errata

```
#include <stdio.h>

int main(void) {
    int limit, num;
    int i, j;
    int part, tot;
    printf("Quanti numeri vuoi inserire? ");
    scanf("%d", limit);          //corretta: scanf("%d", &limit);

    tot = 0;
    for (i=0; i<limit; i++) {
          printf("Inserisci il numero %d:", i+1);
          scanf("%d", &num);
          part = num * num;
          tot = tot + part;
    }
    printf("Totale: %d\n", tot);
    return (0);
}
```

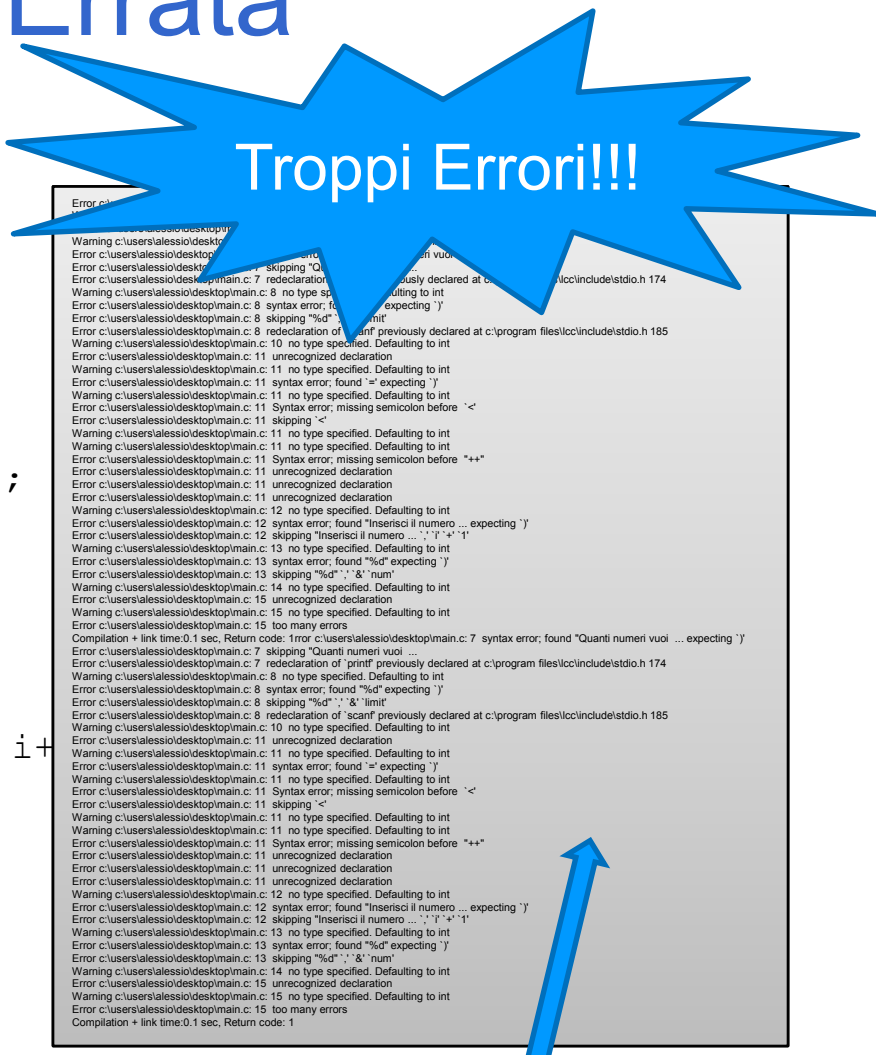
Warning c:\users\alessio\desktop\main.c: 4 **possible usage of limit before definition**
Compilation + link time:0.2 sec, Return code: 0

Versione Errata

```
#include <stdio.h>


int main(void); {
    int limit, num;
    int i, j;
    int part, tot;
    printf("Quanti numeri vuoi inserire? ");
    scanf("%d", &limit);

    tot = 0;
    for (i=0; i<limit; i++) {
          printf("Inserisci il numero %d:", i+
          scanf("%d", &num);
          part = 1;
          for (j=num; j>0; j--)
                    part=part*j;
          tot = tot + part;
```



Il compilatore non sempre indica l'**errore effettivo,** ma ne indica le conseguenze!!

Versione Errata


```
#include <stdio.h>

int main(void) {
    int limit, num;
    int i, j;
    int part, tot;

    printf("Quanti numeri vuoi inserire? ")
    scanf("%d", &limit);

    tot = 0;
    for (i=0; i<limit; i++) {
         printf("Inserisci il numero %d:", i+1);
         part = 1;
         for (j=num; j>0; j--)
                  part=part*j;
         tot = tot + part;
    }
    printf("Totale: %d\n", tot);
    return (0);
}
```

Error c:\users\alessio\desktop\main.c: 8 Syntax error; **missing semicolon before `scanf'**

Versione Errata

```
#include <stdio.h>
int main(void) {
    int limit, num;
    int i, j;
    int part, tot;
    printf("Quanti numeri vuoi inserire? ");
    scanf("%d", &limit);

    tot = 0;
    while (i<limit) {
        printf("Inserisci il numero %d:", i+1);
        scanf("%d", &num);
        part = 1;
        for (j=num; j>0; j--)
            part=part*j;
        tot = tot + part;
    }
    printf("Totale: %d\n", tot);
    return (0);
}
```

Il programma compila, ma l'esecuzione non è corretta!

La variabile '***i***' non è inizializzata => **valore casuale**

Il ciclo dipende dal valore casuale contenuto in '*i*'

Manca l'incremento della variabile contatore ***i++;***

Non sottovalutate i WARNING!

Warning c:\users\alessio\desktop\main.c: 5 **possible usage of i before definition**

Versione Errata

```
#include <stdio.h>

int main(void) {
    int i, j;
    int part, tot;
    printf("Quanti numeri vuoi inserire? ");
    scanf("%d", &limit);

    tot = 0;
    for (i=0; i<limit; i++) {
          printf("Inserisci il numero %d: ", i+1);
          scanf("%d", &num);
          pa
          fo
                  part=part*j;
          tot = tot + part;
    }
    printf("Totale: %d\n", tot);
    return (0);
}
```

Manca la dichiarazione delle variabili ***limit*** e ***num.***

Error c:\users\alessio\desktop\main.c: 8 **undeclared identifier `limit'**
Error c:\users\alessio\desktop\main.c: 14 **undeclared identifier `num'**

Errori Comuni

Ovvero: quello che avreste dovuto sapere (ma che non sapevate) e non avete mai osato chiedere…

Errore

Restituire, tramite una funzione, una stringa allocata come variabile locale (allocata nello *stack*).

Purtroppo, in C, una stringa è solamente un povero array di caratteri. Purtroppo, in C, gli array sono sostanzialmente puntatori costanti ad aree di memoria allocate sullo stack se gli array sono dichiarati come comuni variabili. Purtroppo, come tutte le variabili allocate sullo stack, anche gli array vengono distrutti una volta usciti dallo scope in cui sono stati dichiarati, pertanto una funzione scritta come:

```
char* getString()
{
    char str[20] = “Una stringa”;
    return str;
}
```

Ed utilizzata come:

```
{
    char *stringa;
    stringa = getString();
    printf(“%s”, stringa);
}
```

Non darà luogo a clamorosi errori di compilazione ma solo ad un semplice warning del tipo “attenzione, si sta restituendo l’indirizzo di una variabile locale…”. Peccato che, una volta terminata l’esecuzione della funzione getString, l’area di memoria riservata per l’array di caratteri (char str[20]) viene deallocata ma viene comunque correttamente restituito un puntatore a quell’area di memoria. Tale area di memoria viene poi impropriamente utilizzata successivamente per stampare un dato… che probabilmente sarà ancora lui, ma che sicuramente cambierà molto in fretta, ad esempio alla successiva allocazione di una variabile sullo stack.

Errori Comuni

Soluzione

Per fare le cose in modo corretto, occorre allocare dentro la getString la memoria in modo dinamico in modo da poter restituire l'indirizzo ad un'area di memoria che al termine della getString non venga deallocata. Rovescio della medaglia, una volta che si termina di utilizzare l'area di memoria allocata dalla getString, occorre deallocarla.

```
char* getString()
{
    char *str = (char*)malloc(20 * sizeof(char));
    strcpy(str, "Una stringa");
    return str;
}
```

L'utilizzo…

```
{
char *stringa;
stringa = getString();
printf("%s", stringa);
free(stringa);
    }
```

Errori Comuni

Errore

Copiare le stringhe con l'operatore di assegnamento; confrontare le stringhe con l'operatore di confronto.

Soluzione

Le stringhe sono array. Gli array sono puntatori costanti. Pretendere di copiare un array in un altro utilizzando l'operatore di assegnamento è pretendere di violare il fatto che gli array sono puntatori costanti. In questo caso viene segnalato un bell'errore. Al più è possibile fare in modo che un puntatore a caratteri punti alla stessa area di memoria dell'array: è stato creato un alias della stessa stringa (array) ma non una stringa diversa. Per avere due stringhe diverse l'una copia dell'altra, occorre utilizzare la funzione **strcpy** che si trova nella libreria **string.h**.

Ovviamente, per ciò che è stato detto sopra, non è nemmeno possibile confrontare le stringhe (array) con gli operatori di confronto: si ottiene solamente di confrontare gli indirizzi di memoria dove sono allocate le stringhe e non il loro valore "lessicografico". Per confrontare correttamente due stringhe, utilizzare la funzione **strcmp** che si trova nella libreria **string.h**.

Errori Comuni

Errore

Incapsulare in una struttura una stringa allocata dinamicamente sperando che copiando la struttura venga copiata anche la stringa:

```
typedef struct
{
char *s;
    } myStringType;
```

Soluzione

Quando questa struttura viene copiata, viene copiato solamente il puntatore alla stringa e non viene fatta una copia dell'area di memoria puntata. Tale area è allocata in memoria dinamica e non sullo stack: al termine della copia entrambe le strutture conterranno un puntatore che punterà alla stessa area di memoria… che non è la stessa cosa che fare una copia. Se la struttura contenesse un array allora il discorso sarebbe diverso: in questo caso la struttura contiene direttamente l'area di memoria cui si riferisce l'array quindi copiare la struttura significa copiare anche l'array. La soluzione è: evitare di fare una cosa del genere poiché è del tutto inutile.

Errore

Nella lettura di file di testo, non verificare la terminazione del file.

Soluzione

In un file di testo, la verifica di terminazione del file deve essere fatta in modo opportuno a seconda del metodo di lettura che si sceglie. Se si utilizza la **fscanf**, occorre verificare che il valore di ritorno sia pari al numero di campi che si desiderano leggere (numero di formati**%qualcosa** presenti nella stringa di formato) e se è diverso dal valore atteso, valutare la funzione **feof(fileHandler)** sul file che si sta utilizzando per verificare che non si sia giunti al termine del file (tale funzione restituisce **true** se è stata raggiunta la fine del file). Se si utilizza la **fgetc()**, verificare che il carattere restituito sia diverso dal carattere **EOF** che rappresenta proprio la fine del file.

Errori Comuni

Errore

Allocare la memoria… poi non deallocarla (memory leak!).

Soluzione

Quando si allocano porzioni di memoria dinamica, è necessario ricordarsi di deallocare la memoria una volta terminato il suo utilizzo. In caso contrario è probabile l'applicazione che si sta scrivendo funzioni correttamente nel breve ma che sia estremamente instabile nel lungo termine in quanto occupa memoria senza mai liberarla. In pratica è sempre bene che **malloc** e **free** viaggino in coppia (anche se a distanza…).

Errore

Compilare solo cinque minuti prima della consegna. E' praticamente impossibile consegnare un programma che sia compilabile e che funzioni.

Soluzione

Ogni volta che si scrivono poche righe di codice "consistenti", compilare per verificare che l'applicazione si compili senza errori e senza *warnings*. È un'operazione che porta via pochi secondi ma che risulta essere determinante per un soddisfacente risultato finale.

Errori Comuni

Errore

Eseguire solo cinque minuti prima della consegna. E' praticamente impossibile consegnare un programma che funzioni.

Soluzione

Anche in mancanza di errori e *warnings* è assai probabile inserire nel programma errori algoritmici che possono portare come minimo al non corretto funzionamento del programma e che, normalmente, portano anche instabilità del programma stesso (chiusure inattese). Per evitare ciò, ogni volta che si completa la codifica di una funzionalità o un algoritmo, è bene testare (almeno brevemente, ma sarebbe meglio in modo estensivo…) ciò che si è implementato per verificarne il corretto funzionamento. In generale, nei nostri semplici programmi, occorrono pochi minuti per la verifica del corretto funzionamento.

Errore

Non leggere tutto il testo del compito almeno una volta prima di cominciare a pensare e pretendere di aver capito il contesto del compito.

Soluzione

Leggere tutto il testo del compito almeno una volta prima di cominciare a pensare: a volte i suggerimenti sono in fondo al testo…

Consigli Utili

COMPILARE, COMPILARE, COMPILARE!!

(e durante il compito, inviare i file SPESSO!)

Controllare i **';'** (semicolon)

Controllare le coppie di parentesi ( ) { }

Inizializzate le variabili! Se i valori sono "strani",
state stampando a video variabili non inizializzate!

La procedura scanf richiede l'indirizzo della variabile ➔
&nome_variabile

NON sottovalutare I **WARNING**!!