

Lab. di Sistemi Operativi

Esercitazioni proposte per le lezioni del 22 e 23/04/10

Utilizzando il compilatore gcc in Linux e disponendosi in gruppi di due persone per ogni PC del laboratorio.

1. Utilizzare l'editor vim per realizzare il sorgente in linguaggio "c" di un programma il cui obiettivo sia quello di emettere su stdout il messaggio "Ciao, mondo!". Assegnare a tale file sorgente il nome "es51.c".
2. Utilizzare il gcc per produrre il file eseguibile corrispondente al comportamento descritto dal sorgente es51.c
3. Quale effetto ha il comando gcc invocato con argomento es51.c? Cosa viene prodotto?
4. Che cosa rappresenta il file a.out?
5. Mettete in esecuzione il file a.out prodotto dal compilatore. Visualizzate l'exit value del processo che ha eseguito a.out e, nel caso in cui non abbiate terminato la funzione main con il return di un valore specifico, modificate il sorgente di conseguenza
6. Perfezioniamo l'invocazione del compilatore: il gcc accetta un numero enorme di opzioni che modificano il proprio comportamento. Possiamo, ad esempio, utilizzare l'opzione **-o** seguita da un argomento per indurre il compilatore ad assegnare al file prodotto il nome espresso dall'argomento. Invocare il gcc sul sorgente es51.c in modo che venga generato un file eseguibile di nome **es51**
7. Mettiamo in esecuzione il nuovo file prodotto dal compilatore. Verifichiamo che i caratteri emessi dalla funzione **printf** sono realmente diretti su stdout provando a ridirigere alternativamente stdout e stderr su /dev/null.
8. Introduciamo qualche imperfezione nel file sorgente, ad esempio rimuovendo la direttiva include necessaria per il prototipo della printf. Come reagisce il compilatore? Cambia qualcosa se aggiungiamo l'opzione **-Wall**?
9. Come utilizzare le funzioni di output formattato: stampare il valore di una variabile intera con printf (es52.c).
10. Stampare un singolo carattere (es53.c).
11. Stampare un array di caratteri (es54.c).
12. Array e puntatori: Modificare es54.c in modo da stampare anche il valore dell'array come puntatore.
13. Di che tipo è la variabile array?
14. Possiamo assegnare un valore alla variabile **array**? Proviamo a farlo modificando opportunamente es54.c.
15. Possiamo avere l'value dello stesso tipo di array? Creiamo la variabile a2 di tipo **char ***.
16. a2 e' una copia di array? Se modifico il contenuto di array cosa avviene a a2?
17. Accesso alla linea di comando: Creare un programma che visualizzi il numero di argomenti presenti nella propria command line (es55.c)
18. Modificare es55.c in modo che venga visualizzato il valore di argv.

Soluzione

Esercitazioni proposte per le lezioni del 22 e 23/04/10

Utilizzando il compilatore gcc in Linux e disponendosi in gruppi di due persone per ogni PC del laboratorio.

1. Utilizzare l'editor vim per realizzare il sorgente in linguaggio "c" di un programma il cui obiettivo sia quello di emettere su stdout il messaggio "Ciao, mondo!". Assegnare a tale file sorgente il nome "es51.c".

Soluzione:

```
#include <stdio.h>
int main(int argc, char **argv) {
    printf("Ciao, mondo!\n");
    return(0);
}
```

2. Utilizzare il gcc per produrre il file eseguibile corrispondente al comportamento descritto dal sorgente es51.c

Soluzione:

```
$ gcc es51.c
```

3. Quale effetto ha il comando gcc invocato con argomento es51.c? Cosa viene prodotto?

Soluzione:

```
$ ls -l
...
-rw-r--r-- 1 valealex valealex 94 2006-02-12 21:54 es51.c
-rwxr-xr-x 1 valealex valealex 7053 2006-02-12 21:54 a.out
...
```

4. Che cosa rappresenta il file a.out?

Soluzione:

```
# Il file a.out contiene la sequenza di istruzioni binarie che il
# microprocessore deve eseguire per produrre il comportamento descritto
# dal file sorgente secondo le regole sintattiche del linguaggio c
```

5. Mettete in esecuzione il file a.out prodotto dal compilatore. Visualizzate l'exit value del processo che ha eseguito a.out e, nel caso in cui non abbiate terminato la funzione main con il return di un valore specifico, modificate il sorgente di conseguenza

Soluzione:

```
$ ./a.out
$ echo $?
0
```

6. Perfezioniamo l'invocazione del compilatore: il gcc accetta un numero enorme di opzioni che modificano il proprio comportamento. Possiamo, ad esempio, utilizzare l'opzione **-o** seguita da un argomento per indurre il compilatore ad assegnare al file prodotto il nome espresso dall'argomento. Invocare il gcc sul sorgente es51.c in modo che venga generato un file eseguibile di nome **es51**

Soluzione:

```
$ gcc es51.c -o es51
$ ls -l
```

7. Mettiamo in esecuzione il nuovo file prodotto dal compilatore. Verifichiamo che i caratteri emessi dalla funzione **printf** sono realmente diretti su stdout provando a ridirigere alternativamente stdout e stderr su /dev/null.

Soluzione:

```
$ ./es51 > /dev/null
$ ./es51 2> /dev/null
Ciao, mondo!
```

8. Introduciamo qualche imperfezione nel file sorgente, ad esempio rimuovendo la direttiva include necessaria per il prototipo della printf. Come reagisce il compilatore? Cambia qualcosa se aggiungiamo l'opzione **-Wall**?

Soluzione:

```
$ gcc es51.c -o es51
$ gcc es51.c -o es51 -Wall
# Nella seconda invocazione, si richiede al gcc l'emissione dei
# messaggi di warning (avvertimento) legati al controllo sintattico
# del sorgente. Omettendo l'include, il gcc segnala l'assenza del
# prototipo della funzione printf, effettivamente incluso nel file
# stdio.h
```

9. Come utilizzare le funzioni di output formattato: stampare il valore di una variabile intera con printf (es52.c).

Soluzione:

```
/* file: es52.c */
/* job: stampa il valore di i, intero */
/* compilare con: gcc -Wall es52.c -o es52 */
#include <stdio.h>
int i; /* variabile globale, dichiarazione */
int main(int argc, char **argv) {
    i=0; /* assegnamento */
    printf("i vale: %d\n", i);
    return(0);
}
```

10. Stampare un singolo carattere (es53.c).

Soluzione:

```
/* file: es53.c */
/* job: stampa il valore di c, carattere */
/* compilare con: gcc -Wall es53.c -o es53 */
#include <stdio.h>
char c; /* var. globale, dichiarazione */
int main(int argc, char **argv) {
    c='A'; /* assegnamento */
    printf("c vale: %c\n", c);
    return(0);
}
```

11. Stampare un array di caratteri (es54.c).

Soluzione:

```
/* file: es54.c */
/* job: stampa un array di char */
#include <stdio.h>
char array[50]; /* dichiarazione, riserva
               lo spazio necessario in memoria */
int main(int argc, char **argv) {
    array[0]='A'; /* assegnamento al
                 primo elemento */
    array[1]='\0'; /* secondo elemento,
                  carattere nullo che rappresenta
                  il fine stringa */
    printf("array_vale_%s\n", array);
    return(0);
}
```

12. Array e puntatori: Modificare es54.c in modo da stampare anche il valore dell'array come puntatore.

Soluzione:

```
/* file: es54.c */
/* job: stampa un array di char */
#include <stdio.h>
char array[50]; /* dichiarazione, riserva
               lo spazio necessario in memoria */
int main(int argc, char **argv) {
    array[0]='A'; /* assegnamento al
                 primo elemento */
    array[1]='\0'; /* secondo elemento,
                  carattere nullo che rappresenta
                  il fine stringa */
    printf("array_vale_%s\n", array);
    printf("array_vale_%p\n", array);
    return(0);
}
```

13. Di che tipo è la variabile array?

Soluzione:

```
# Come ogni vettore di elementi, la variabile
# array e' di tipo "puntatore" quindi rappresenta
# un indirizzo in memoria. Con maggiore attenzione,
# possiamo affermare che array e' un puntatore a
# char (il tipo degli elementi contenuti nel
# vettore) che punta alla locazione occupata dal
# primo elemento.
```

14. Possiamo assegnare un valore alla variabile **array**? Proviamo a farlo modificando opportunamente es54.c.

Soluzione:

```

/* file: es54.c */
/* job: stampa un array di char */
#include <stdio.h>
char array[50]; /* dichiarazione, riserva
                lo spazio necessario in memoria */
int main(int argc, char **argv) {
    array[0]='A'; /* assegnamento al
                  primo elemento */
    array[1]='\0'; /* secondo elemento,
                  carattere nullo che rappresenta
                  il fine stringa */
    printf("array_vale %s\n", array);
    printf("array_vale %p\n", array);
    array=(char*)0; /* errore: array non lvalue */
    /* un lvalue e' qualcosa che puo' trovarsi alla sinistra
       di un assegnamento (lvalue=left value). array e' un
       puntatore ma di valore costante in quanto punta e puntera' sempre
       alla prima locazione dell'area di memoria riservata all'atto
       della sua dichiarazione. Per il compilatore array e' assimilabile
       ad un simbolo, quindi ad una costante. */
    printf("array_vale %p\n", array);
    return(0);
}

```

15. Possiamo avere lvalue dello stesso tipo di array? Creiamo la variabile a2 di tipo **char ***.

Soluzione:

```

/* file: es54.c */
/* job: stampa un array di char */
#include <stdio.h>
char array[50]; /* dichiarazione, riserva
                lo spazio necessario in memoria */
char *a2; /* dichiarazione, riserva lo
           spazio per un puntatore */
int main(int argc, char **argv) {
    array[0]='A';
    array[1]='\0';
    a2=array; /* corretto: a2, variabile dello stesso
              tipo di array, viene resa uguale ad array */
    printf("a2_vale %s\n", a2);
    printf("a2_vale %p\n", a2);
    a2=(char*)0; /* corretto: a2 e' lvalue */
    printf("a2_vale %p\n", a2);
    return(0);
}

```

16. a2 e' una copia di array? Se modifico il contenuto di array cosa avviene a a2?

Soluzione:

```

# a2 e' un secondo riferimento alla memoria puntata da array, quindi
# sia a2 che array fanno riferimento alla stessa porzione di memoria.
# a2, quindi, non e' una copia dell'intero vettore ma solo un nuovo
# riferimento. Se modifico array[2], anche a2[2] sara' modificato.

```

17. Accesso alla linea di comando: Creare un programma che visualizzi il numero di argomenti presenti nella propria command line (es55.c)

Soluzione:

```
/* file: es55.c */
/* job: accesso alla command line */
#include <stdio.h>
int main(int argc, char **argv) {
    printf(" Il valore di argc e': %d\n", argc);
    return(0);
}
/* L'interfaccia della funzione main() rispecchia
 * quella dei processi unix: in ingresso troviamo
 * le informazioni (argc, argv) necessarie per accedere
 * alla command line ed in uscita possiamo ritornare
 * un numero intero (exit value) */
```

18. Modificare es55.c in modo che venga visualizzato il valore di argv.

Soluzione:

```
/* file: es55.c */
/* job: accesso alla command line */
#include <stdio.h>
int main(int argc, char **argv) {
    printf(" Il valore di argc e': %d\n", argc);
    printf(" Il valore di argv e': %p\n", argv);
    /* argv e' una variabile di tipo puntatore,
     * il tipo degli elementi cui punta e' "char*",
     * quindi argv e' un puntatore a un puntatore
     * a carattere. argv[0] e' il primo elemento
     * (puntatore a carattere) del vettore argv */
    printf(" Il valore di argv[0] e': %p\n", argv[0]);
    /* argv[0] e' un puntatore a carattere, quindi
     * potrebbe essere una stringa c */
    printf(" Il valore di argv[0], come stringa, e': %s\n", argv[0]);
    /* argv[0] e' la prima stringa della command line;
     * in generale argv[i] e' la i-esima stringa della
     * command line a patto che i sia minore di argc */
    return(0);
}
```