4. QUARTA esercitazione in laboratorio

Attivare il TC e posizionarsi sul proprio dischetto. Ciò può esser fatto usando il comando **Change Dir** del menù **File**, oppure attivando il comando **DOS shell** (stesso menù), spostandosi su A: e inserendo EXIT per tornare al TC.

Le soluzioni proposte durante l'esercitazioni (indicate spesso tra parentesi nei titoli delle varie sezioni) sono reperibili via web, tramite il **sito del cors**o (zona "esercitazioni autoguidate")

4.1. I maggiori (`FUNMAX.C`)

Scrivere un programma C che legge due numeri interi e stampa il più grande dei due. Però il programma deve far uso di una funzione `maggiore()`, definita dal programmatore, che riceva due parametri interi e restituisca il valore maggiore tra i due parametri.

4.2. I minori (`FUNMIN.C`)

Scrivere un programma C che legge due numeri interi e stampa il più piccolo dei due. Però il programma deve far uso di una funzione `minore()`, definita dal programmatore, che riceva due parametri interi e restituisca il valore maggiore tra i due parametri.

Nel programma proposto come soluzione vengono messe in pratica due azioni significative:

1) la funzione minore restituisce il risultato usando una tra due distinte `return`:

```
if (primo < secondo)
   return (primo);
else return(secondo);
```

   Se non lo si è già fatto, provare questa soluzione, eseguendo il programma passo-passo e verificando come, ovviamente, una sola delle due `return` venga eseguita (e quando viene eseguita, l'esecuzione della funzione termina e si torna sulle istruzioni della `main`).

2) la `printf` che stampa, nella `main`, il minore tra i due input, stampa direttamente il risultato della chiamata della funzione minore() (senza dover usare variabili di appoggio come la `m` di `FUNMAX.C`).

```
printf("il...tra i due e' %d\n", minore(num1,num2) );
```

4.3. ESECUZIONE PASSO-PASSO CON F8

Finora siamo stati capaci di eseguire programmi "istruzione per istruzione" usando F7. La funzione espletata da F7 è chiamata *trace*, in quanto permette di "tracciare" l'esecuzione del programma, normalmente tenendo sotto controllo il contenuto di tutte o alcune variabili. Se, quando siamo posizionati su una linea del programma, premiamo F7, l'istruzione corrispondente viene eseguita e ne possiamo vedere gli effetti nella finestra di watches. Se l'istruzione era una chiamata di funzione, incominciamo a tracciare anche le istruzioni della funzione chiamata, per poi tornare alle istruzioni dell'unità chiamante, quando quella chiamata è terminata.

In TC c'è anche un'altra funzione analoga: *step over*, attivata dal tasto F8. Premendo il tasto F8 quando siamo posizionati su una linea del programma che contiene una chiamata, la chiamata viene eseguita direttamente, senza tracciare le istruzioni della funzione chiamata.

Sperimentare questa funzionalità sui programmi precedenti: eseguirli passo passo, usando F8 anziché F7. L'unica cosa che cambia è che non vediamo passo-passo l'esecuzione delle chiamate di `minore()` o `maggiore()`. Possiamo usare questa modalità quando una funzione ha un;'esecuzione molto lunga e non intendiamo vedere approfonditamente l'esecuzioned elle sue istruzioni.

4.4. UNITÀ CHIAMANTI e UNITÀ CHIAMATE (FUNPARZ.C)

Ogni funzione può essere chiamante (se chiama un'altra funzione) e/o chiamata (se viene chiamata da qualche altra funzione).

Scrivere un programma che legge una sequenza di 6 numeri interi e stampa il più grande. Il programma però deve essere costruito come segue:

- deve essere definita una funzione `maggiore()` che, ricevendo due parametri interi, restituisce il maggiore;
- deve esserepoi definita una funzione `maxSequenza()` che esegue la lettura di 6 numeri e restituisce il massimo tra essi riscontrato; si tratta di una funzione senza parametri, che chiama `maggiore()` per confrontare ogni numero letto con un massimo parziale, secondo l'algoritmo noto;
- infine deve essere definita la main() che chiama maxSequenza() e ne usa il risultato per stampare qual è stato il numero massimo incontrato.
-

In altre parole, ecco uno schema del programma complessivo:

```
#include<stdio.h>
#define QUANTI_NUMERI 6
... definizione di int maggiore(int primo, int secondo)...
... definizione di int maxSequenza()...
int main ()
{
...
  massimo=maxSequenza();

  printf("---- il massimo numero dato e' %d\n", massimo );

...
return 0;
}
```

4.5. ANCORA UNITÀ CHIAMANTI e UNITÀ CHIAMATE (FUNMINPZ.C)

Scrivere un programma che legge una sequenza di `n` valori interi, con `n` dato in input, e stampa il vaolore minimo riscontrato. Il programma però deve essere costruito come segue:

- deve essere definita una funzione `minore()` come sopra;
- deve essere poi definita una funzione `minSeq(int numeroInput)` che esegue la lettura di `numeroInput` numeri interi e ne restituisce il minimo;
- infine deve essere definita la `main()` che legge `n` e chiama `minSeq()`.

4.6. CALL STACK

Come visto nell'esercizio precedente, durante l'esecuzione di una chiamata può darsi che venga attivata un'altra chiamata. Ad esempio durante l'esecuzione della `main()` viene attivata `minSeq(...)` con certi parametri, e durante l'esecuzione di questa viene chiamata `minore()`.

Il comando **Call Stack** del menù ***Debug*** permette di vedere lo stato delle chiamate di funzione. La sequenza di chiamate ancora non terminate viene mostrata in ordine inverso (in modo che sulla linea più alta sia indicata la funzione effettivamente al momento attiva. Per ogni chiamata ancora attiva vengono indicati il nome della funzione e i parametri della chiamata relativa.

4.7. Tabella Celsius/Fahreneit/Kelvin (TABELLA3.C)

Rieseguire l'esercizio 3.11, in modo che i valori Fahreneit e Kelvin, relativi ad un dato Celsius, vengano calcolati da una opportuna funzioni (ad es. `Fahr()` e `Kelv()`).

4.8. Funzione per il massimo comun divisore (FUNMCD.C)

Scrivere un programma che legge varie coppie di numeri interi e stampa per ognuna il relativo massimo comun divisore. La funzione `main()` deve usare una funzione `mcd()` che, ricevuti due numeri interi, restituisca il relativo mcd. Il programma termina quando almeno uno dei numeri in una coppia è 0 (zero).

4.9. Esperimento sulla visibilità degli identificatori

Nell'esercizio precedente sul massimo comun divisore la soluzione proposta in `FUNMCD.C` ha la seguente struttura:

```
int main ()
{
  int primo,secondo;     /* i numeri di cui trovare il MCD */
...
  while ( (primo!=0) && (secondo!=0) ) {
    printf("mcd tra %d e %d e' %d\n", primo,secondo, mcd(primo,secondo));
...
printf("FINE\n");
return 0;
}
...
int mcd(int n, int m) {

  while (n!=m)
...
  return n;       /* o m ... */
}
```

Provare a sostituire la printf segnalata sopra con la seguente:

```
    printf("mcd tra %d e %d e' %d\n", n,m, mcd(n,m));
```

Cosa succede?

> Succede che i simboli `n` ed `m` **non** sono noti nel blocco di istruzioni della `main()` e quindi risultano *undefined*. È ovvio che sia così! L'unico blocco in cui `n` ed `m` sono noti è quello della funzione `mcd()`, in cui questi simboli identificano dei parametri.

4.10. Esperimento sul "passaggio di parametri" ad una funzione

Sempre facendo riferimento alla soluzione proposta in `FUNMCD.C`, provare a far girare il programma proposto, mediante un'esecuzione passo passo (con F7, in modo da tracciare anche il comportamento delle chiamate a `mcd()`. Durante queste esecuzioni, tenere sotto controllo gli identificatori `primo`, `secondo` `n` ed `m`.
Ovviamente `n` ed `m` risultano esistere solo mentre stiamo eseguendo la chiamata ad `mcd()` e invece sono indefiniti mentre stiamo eseguendo le istruzioni di altre funzioni, come la `main()`). E analogamente, `primo` e `secondo` sono visibili solo mentre stiamo eseguendo istruzioni della funzione in cui sono definiti (sempre la `main()`).

Comunque, se tutte queste quattro locazioni sono elencate tra le Watches, possiamo renderci conto che durante l'esecuzione di una chiamata di `mcd(primo, secondo)`

- `n` ed `m` inizialmente hanno i medesimi valori di `primo` e `secondo`,
- poi `n` ed `m` cambiano progressivamente fino a diventare uguali
- ma, quando la chiamata è terminata, `primo` e `secondo` non sono cambiati, cioè hanno conservato il valore che avevano prima della chiamata. Cioè i cambiamenti di n ed m non s i sono riflessi su primo e secondo.

Il fatto che `primo` e `secondo` (i parametri attuali della chiamata di `mcd()`) mantengano il loro valore (e non cambino come fanno `n` ed `m`) è una conseguenza delle modalità del passaggio dei parametri nelle funzioni C.
Quando `mcd(primo, secondo)` viene attivata, nel record di attivazione vengono allocate due locazioni per `n` ed `m` (i parametri formali). In queste locazioni vengono copiati i valori dei parametri attuali `primo` e `secondo`.

Ogni uso che facciamo di `n` ed `m` nella funzione `mcd()` si riflette sulle locazioni allocate per loro nel recod di attivazione e non su quelle dei parametri attuali.

4.11. Funzione per la media

Scrivere un programma che esegua varie volte le seguenti operazioni:

1) lettura di un numero `n`
2) lettura di n numeri interi e calcolo della relativa media
3) stampa della mediacalcolata al punto 2)

L'operazione di cui al punto 2) deve essere eseguita da una funzione opportuna. Il programma termina quando viene letto un valore 0 per `n`.

4.12. Funzione per la media (2)

Come sopra, ma facendo in modo che la funzione `main()` stampi, alla fine, il valore massimo riscontrato tra tutte le medie.

4.13. Schema di scambio (`SCAMBIO.C`)

Per introdurre il discorso sugli effetti collaterali su parametri (e, prima ancora, su variabili *top level*), scriviamo un programma che legge due numeri in due variabili, stampa le due variabili, le scambia tra loro e poi le ristampa.

4.14. Variabili TOP LEVEL (`TOPLEVEL.C`)

È top level qualunque dichiarazione o definizione posta in un file al di fuori di funzioni. Ad esempio, nel file `TOPLEVEL.C` la variabile `massimo` è definita "top level". Anche la dichiarazione della funzione `maggiore()` è top level.

Le entità top level si dicono anche *globali*. Esse sono visibili da qualunque funzione definita dopo di loro. Ecco perché la funzione `main()` può usare la variabile `massimo` anche se questa non è definita nel suo blocco. Analogamente anche la funzione `maggiore()` potrebbe usare `massimo` (ma non lo abbiamo fatto).

Verificare il funzionamento del programma proposto, monitorando il contenuto delle variabili con delle watches.

Poi modificare il programma, facendo in modo che sia `maggiore()` stessa a modificare la variabile globale massimo (mentre `main()` usa `massimo` solo per stamparne il valore. Soluzione in `TOPLEV2.C`. (Suggerimento: `maggiore()`mette il più grande tra i suoi due parametri in `massimo`. Quindi non c'è bisogno che ritorni risultati. Perciò può benissimo essere una funzione void. Ecco il prototipo: `void maggiore(int, int);`)

4.15. Esercizio: occorrenze di dati in un intervallo (`OCCORR.C`)

Scrivere un programma tale che
INPUT:
- due interi che rappresentano gli estremi di un intervallo: `inf` e `sup`;
- legge N numeri interi (N è un simbolo costante)

OUTPUT:
- quanti degli N numeri interi erano compresi nell'intervallo (estremi compresi).

Realizzare il programma secondo queste specifiche:
- ci sono due variabili top level, `inf` e `sup`;
- la funzione `main()`legge i valori per `inf` e `sup` e li scambia, se l'utente non li ha forniti in ordine;
- la funzione `main()`legge N interi e usa un contatore `aux` per tener traccia di quelli compresi nell'intervallo;
- c'è una funzione `swapSupInf()` che si occupa dello scambio tra `inf` e `sup` (quando serve, viene chiamata da `main()`);
- la funzione `swapSupInf()` usa una variabile di appoggio `aux` per effettuare lo scambio tra `inf` e `sup` (e non ci sono confusioni con la `aux` di `main()`!!!).

4.16. La funzione scambiosb() che non funziona bene (`FUNSCSB.C`)

Nel file `FUNSCSB.C` abbiamo provato a scrivere una funzione `scambiosb()` che riceva due parametri e ne scambi contenuti. la soluzione non funziona, per il caratteristico metodo di passaggio dei parametri in C: dato che i parametri formali sono locazioni allocate nel record di attivazione, riempite con copie dei valori contenuti nei parametri attuali, questi ultimi (cioè le variabili `num1` e `num2`) non vengono effettivamente scambiati. Esaminare il programma, eseguirlo passo-passo, monitorando le variabili coinvolte, in modo da rendersi bene conto che le cose non possono funzionare così.

Per fare in modo che la funzione di scambio riesca effettivamente a modificare il contenuto delle locazioni associate ai parametri attuali bisogna che questi siano in realtà indirizzi (gli indirizzi delle locazioni che l'unità chiamante vuole effettivamente vengano modificate).

Basandosi su quanto chiarito a lezione e sul libro, provare a scrivere una nuova edizione del programma precedente, in cui agisca una funzione `scambio()` che, ricevuti due indirizzi di locazioni, scambia il contenuto di tali locazioni usando l'operatore di indirezionamento `*`.

Il programma legge due variabili e chiama la funzione di scambio passandole gli indirizzi delle due variabili. Dopo la chiamata della funzione di scambio, il programma stampa le due variabili, verificando che i loro valori siano stati effettivamente scambiati. Una soluzione corretta è nel file `FUNSCAMB.C`.

4.17. Esercizio (`CONTR1.C`)

Bisogna assolutamente scivere un programma che
- legge N caratteri da input e

- calcola e stampa
  - quello di codice ascii maggiore e
  - quello di codice ascii minore.
  -

N e' una costante definita ad esempio come 10.
Il programma legge la sequenza di caratteri e deve usare una funzione `controlla()` che viene chiamata come segue, con i parametri attuali specificati:

```
controlla(car, carmax, carmin, &flagmax, &flagmin);
```

dove `ch` e' uno dei caratteri letti da tastiera, `carmax` e `carmin` sono il massimo e minimo (parziali) carattere letto finora e `flagmax` e `flagmin` sono variabili intere.

L'esecuzione della chiamata deve essere tale da controllare se `ch` e' un nuovo massimo parziale e/o un nuovo minimo parziale;

- se ch e' un nuovo massimo parziale, la locazione associata al parametro attuale `flagmax` deve essere assegnata con il valore 1 (altrimenti con 0);
- se ch e' un nuovo minimo parziale, la locazione associata al parametro attuale `flagmin` deve essere assegnata con il valore 1 (altrimenti con 0).

Successivamente alla chiamata, il programma modifica opportunamente `carmax` e `carmin` (se e' il caso) in modo che queste variabili contengano sempre il massimo e minimo carattere tra quelli letti fino al momento.

4.18. Esercizio (CONTR2.C)

Stesso problema precedente, ma adesso bisogna risolverlo con una funzione `controlla()` definita con meno parametri. Le chiamate che la main() effettua sono del tipo

```
controlla(car, &carmax, &carmin);
```

dove `ch` e' sempre uno dei caratteri letti da tastiera, e `carmax` e `carmin` sono il massimo e minimo (parziali) carattere letto finora.

L'esecuzione della chiamata deve essere tale da controllare se `ch` e' un nuovo massimo parziale e/o un nuovo minimo parziale;

- se ch e' un nuovo massimo parziale, la locazione associata al parametro attuale `carmax` deve essere assegnata con il carattere contenuto in ch (altrimenti non si fa nulla);
- se ch e' un nuovo minimo parziale, la locazione associata al parametro attuale `carmin` deve essere assegnata con il carattere contenuto in ch (altrimenti non si fa nulla).

In questo modo, successivamente alla chiamata, il programma trova massimo e minimo parziale già sistemati e quindi puo' proseguire con le letture successive senza altro da fare.