Capitolo 1 – Introduzione

Outline

Storia del C
C standard library
Vantaggi
Un semplice programma C: Stampa di una linea di testo
Un altro semplice programma C: Sommare due interi
Concetti sulla Memoria
Aritmetica in C
Decisioni: Uguaglianza e Operatori Relazionali
Tipi di dati fondamentali

Storia del C

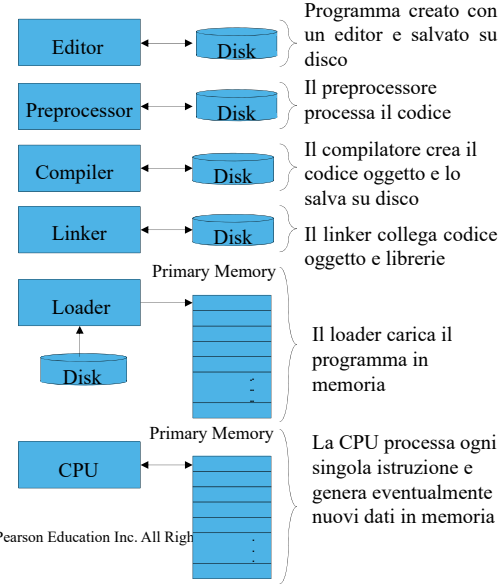
- C
  - Evoluzione di due precedenti linguaggi di programmazione, BCPL e B (by Ritchie)
  - Usato per sviluppare il sistema operativo UNIX
  - Usato per scrivere i moderni sistemi operativi
  - Hardware independent (portabile)
  - Dal tardo 1970 il C si è evoluto nel "Traditional C"
- Standardizzazione
  - Esistono diverse varianti del C, ma non erano compatibili
  - E' stato formato un comitato per creare una definizione "non ambigua, e machine-independent"
  - Standard creato nel 1989, aggiornato nel 1999

C Standard Library

- I programmi C consistono di moduli chiamati funzioni (functions)
  - Un programmatore può creare delle proprie funzioni
    - Vantaggio: il programmatore sa esattamente come funzionano
    - Svantaggio: molto tempo per lo sviluppo
  - I programmatori spesso usano le librerie di funzioni del C
    - Usano queste funzioni come building blocks
  - Riuso: Evitare di re-inventare la ruota
    - Se una funzione esiste già, generalmente è meglio usarla piuttosto che scriverne una propria
    - Le funzioni di libreria sono ben scritte, efficienti e portabili

Base di un tipico ambiente per lo sviluppo di un programma C

- Fasi:
  1. *Edit*
  2. *Preprocess*
  3. *Compile*
  4. *Link*
  5. *Load*
  6. *Execute*



Vantaggi

- C è piccolo
- Descrivibile in poco tempo
- Abbastanza semplice da imparare
- E' efficiente
- Caratterizzato da un buon livello di portabilità

Introduzione

- Linguaggio C
  - Strutturato
- Programmazione strutturata
  - Principi costantemente utilizzati nello sviluppo di tutti i programmi

Un semplice programma C: Stampare una linea di testo

```
1   /* Fig. 2.1: fig02_01.c
2      A first program in C */
3   #include <stdio.h>
4
5   /* function main begins program execution */
6   int main()
7   {
8      printf( "Welcome to C!\n" );
9
10     return 0; /* indicate that program ended successfully */
11
12  } /* end function main */
```

```
Welcome to C!
```

Commenti

- Il testo racchiuso da `/*` e `*/` è ignorato
- Usato per descrivere il programma

- `#include <stdio.h>`
  - Direttiva del preprocessore
    - Indica di caricare il contenuto di un certo file
  - `<stdio.h>` permette operazioni standard di input/output

Un semplice programma C: Stampare una linea di testo

- `int main()`
  - I programmi C contengono una o più funzioni, una delle quali **deve essere** `main`
  - Le parentesi sono usate per indicare una funzione
  - `int` significa che `main` "restituisce" un valore intero
  - Le parentesi graffe (`{` e `}`) indicano un blocco
    - Il corpo di tutte le funzioni deve essere contenuto tra parentesi graffe

Un semplice programma C: Stampare una linea di testo

- `printf( "Welcome to C!\n" );`
  - Istruzione per eseguire una certa azione
    - Stampa la stringa di caratteri all'interno delle virgolette (" ")
  - L'intera linea è chiamata statement
    - Tutti gli statements devono terminare con un punto e virgola (;)
  - Caratteri di escape (\)
    - Indicano che la printf deve fare qualcosa fuori dall'ordinario
    - `\n` è un carattere di nuova linea

Un semplice programma C: Stampare una linea di testo

Sequenza di Escape	Descrizione
\n	Newline. Posiziona il cursore all'inizio della prossima linea.
\t	Horizontal tab. Muove il cursore al prossimo tab stop.
\a	Alert. Riproduce un segnale acustico.
\\	Backslash. Inserisce un carattere di backslash in una stringa.
\"	Double quote. Inserisce delle virgolette in una stringa.

Fig. 2.2 Alcune comuni sequenze di escape.

Un semplice programma C: Stampare una linea di testo

- `return 0;`
  - Un modo per uscire da una funzione
  - `return 0`, in questo caso significa che il programma è terminato normalmente
- Parentesi destra `}`
  - Indica che la fine del `main` è stata raggiunta
- Linker
  - Quando una function viene chiamata, il linker la localizza nella libreria
  - La inserisce nel programma oggetto
  - Se il nome della funzione è scritto in modo errato, il linker produrrà un errore perchè non sarà in grado di trovare la funzione nella libreria


```
1  /* Fig. 2.3: fig02_03.c
2     Printing on one line with two printf statements */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     printf( "Welcome " );
9     printf( "to C!\n" );
10
11    return 0; /* indicate that program ended successfully */
12
13 } /* end function main */
```

fig02_03.c

```
Welcome to C!
```

Program Output

```c
/* Fig. 2.4: fig02_04.c
   Printing multiple lines with a single printf */
#include <stdio.h>

/* function main begins program execution */
int main()
{
   printf( "Welcome\nto\nC!\n" );

   return 0; /* indicate that program ended successfully */

} /* end function main */
```

fig02_04.c

```
Welcome
to
C!
```

Program Output


```c
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

/* function main begins program execution */
int main()
{
   int integer1; /* first number to be input by user  */
   int integer2; /* second number to be input by user */
   int sum;      /* variable in which sum will be stored */

   printf( "Enter first integer\n" );  /* prompt */
   scanf( "%d", &integer1 );           /* read an integer */

   printf( "Enter second integer\n" ); /* prompt */
   scanf( "%d", &integer2 );           /* read an integer */

   sum = integer1 + integer2;          /* assign total to sum */

   printf( "Sum is %d\n", sum );       /* print sum */

   return 0;  /* indicate that program ended successfully */

} /* end function main */
```

fig02_05.c


```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Program Output

Un altro semplice programma C: Sommare due interi

- Come prima
  - Commenti, `#include <stdio.h>` e `main`
- `int integer1, integer2, sum;`
  - Definizione di variabili
    - Variabili: locazioni in memoria dove è possibile memorizzare un valore
  - `int` significa che le variabili possono contenere interi (`-1`, `3`, `0`, `47`)
  - Nomi di variabili (identificatori)
    - `integer1`, `integer2`, `sum`
    - Identificatori: consistono di lettere, cifre (non possono cominciare con una cifra) e underscore (`_`)
      - Case sensitive
  - Le definizioni appaiono prima degli statement che le utilizzano
    - Se uno statement referenzia una variabile non dichiarata, sarà prodotto un errore sintattico da parte del compilatore

Un altro semplice programma C: Sommare due interi

- `scanf( "%d", &integer1 );`
  - Ottiene un valore dall'utente
    - `scanf` usa lo standard input (generalmente la tastiera)
  - `scanf` ha due argomenti (parametri)
    - `%d` – indica che il dato dovrebbe essere un intero decimale
    - `&integer1` - locazione in memoria per memorizzare la variable
    - `&` inizialmente provoca confusione – per ora, ricordate solo di includerlo per i nomi delle variabili nello statement `scanf`
  - Durante l'esecuzione del programma l'utente risponde alla `scanf` inserendo un numero, e poi premendo il tasto *enter* (return)

Un altro semplice programma C: Sommare due interi

- `=` (operatore di assegnamento)
  - Assegna un valore ad una variabile
  - E' un operatore binario (ha due operandi)

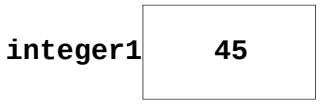
```
sum = variable1 + variable2;
sum avrà variable1 + variable2;
```

  - La variable a sinistra riceve il valore
- `printf( "Sum is %d\n", sum );`
  - Simile alla `scanf`
    - `%d` indica che un intero decimale sarà stampato
    - `sum` specifica quale intero sarà stampato
  - I calcoli posso essere eseguiti all'interno della `printf`

```
printf( "Sum is %d\n", integer1 + integer2 );
```

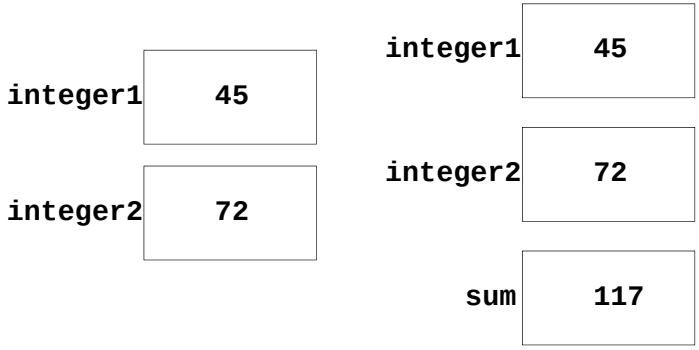

Concetti sulla Memoria

- Variabili
  - I nomi di variabile corrispondono a locazioni di memoria del computer
  - Ogni nome di variabile ha *nome*, *tipo*, *dimensione* e *valore*
  - Ogni volta che un nuovo valore è inserito in una variabile (attraverso una `scanf`, per esempio), esso rimpiazza (e distrugge) il valore precedente
  - La lettura delle variabili dalla memoria non ne cambia il valore
- Una rappresentazione visuale



Concetti sulla Memoria

- Una rappresentazione visuale (continua)



Aritmetica

- Calcoli aritmetici
  - `*` per la moltiplicazione e `/` per la divisione
  - La divisione intera tronca il resto
    - `7 / 5` ritorna `1`
  - Operatore modulo (%) ritorna il resto
    - `7 % 5` ritorna `2`
- Precedenza tra operatori
  - Alcuni operatori hanno priorità maggiore (i.e., moltiplicazione prima della somma)
    - Usa le parentesi quando necessario
  - Esempio: Media tra tre variabili `a`, `b`, `c`
    - Non usare: `a + b + c / 3`
    - Usare: `(a + b + c ) / 3`

Aritmetica

- Operatori aritmetici:

Operazione C	Operatore aritmetico	Espressione algebrica	Espressione in C
Somma	`+`	$f + 7$	`f + 7`
Sottrazione	`-`	$p - c$	`p - c`
Moltiplicazione	`*`	bm	`b * m`
Divisione	`/`	x / y	`x / y`
Modulo	`%`	$r \bmod s$	`r % s`

- Regole di precedenza:

Operatori	Operazioni	Ordine di valutazione (precedenza)
`()`	Parentesi	Valutate per prime. Valgono le regole di innesto delle parentesi. Nel caso di più parentesi allo stesso livello, la valutazione avviene da sinistra a destra.
`*`, `/`, or `%`	Moltiplicazione,Divisione, Modulo	Valutate per seconde. Più operazioni allo stesso livello valutate da sinistra a destra.
`+` o `-`	Somma Sottrazione	Valutate per ultime. Più operazioni allo stesso livello valutate da sinistra a destra.

Decisioni: Uguaglianza e Operatori Relazionali

Step 1. `y = 2 * 5 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`2 * 5 is 10`

Step 2. `y = 10 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`10 * 5 is 50`

Step 3. `y = 50 + 3 * 5 + 7;` *(Multiplication before addition)*

`3 * 5 is 15`

Step 4. `y = 50 + 15 + 7;` *(Leftmost addition)*

`50 + 15 is 65`

Step 5. `y = 65 + 7;` *(Last addition)*

`65 + 7 is 72`

Step 6. `y = 72;` *(Last operation—place 72 in y)*

Decisioni: Uguaglianza e Operatori Relazionali

- Executable statements
  - Eseguono azioni (calcoli, input/output di dati)
  - Eseguono operazioni in base a delle decisioni
    - Stampare `"pass"` o `"fail"` in base al valore di un test
- `if` control statement
  - Versione semplice in questa sezione, più dettagliato in seguito
  - Se una condition è vera (`true`), allora il corpo dello statement `if` viene eseguito
    - `0` è falso (`false`), non-zero è vero (`true`)
  - Il controllo riprende sempre dopo la struttura `if`
- Parole chiave
  - Parole speciali riservate per il C
  - Non possono essere usate come identificatori o nomi di variabili

Decisioni: Uguaglianza e Operatori Relazionali

Operatori standard di uguaglianza e operatori relazionali	Operatori di uguaglianza o relazionali in C	Esempi di condizioni in C	Significato delle condizioni in C
Operatori di uguaglianza			
=	==	x == y	x è uguale a y
•	!=	x != y	x è diverso da y
Operatori Relazionali			
>	>	x > y	x è maggiore di y
<	<	x < y	x è minore di y
> =	> =	x > = y	x è maggiore o uguale a y
< =	< =	x < = y	x è minore o uguale a y


```
1  /* Fig. 2.13: fig02_13.c
2     Using if statements, relational
3     operators, and equality operators */
4  #include <stdio.h>
5
6  /* function main begins program execution */
7  int main()
8  {
9     int num1; /* first number to be read from user  */
10    int num2; /* second number to be read from user */
11
12    printf( "Enter two integers, and I will tell you\n");
13    printf( "the relationships they satisfy: " );
14
15    scanf( "%d%d", &num1, &num2 ); /* read two integers */
16
17    if ( num1 == num2 ) {
18       printf( "%d is equal to %d\n', num1, num2 );
19    } /* end if */
20
21    if ( num1 != num2 ) {
22       printf( "%d is not equal to %d\n", num1, num2 );
23    } /* end if */
24
```

fig02_13.c (Part 1 of 2)


```
25    if ( num1 < num2 ) {
26       printf( "%d is less than %d\n", num1, num2 );
27    } /* end if */
28
29    if ( num1 > num2 ) {
30       printf( "%d is greater than %d\n", num1, num2 );
31    } /* end if */
32
33    if ( num1 <= num2 ) {
34       printf( "%d is less than or equal to %d\n", num1, num2 );
35    } /* end if */
36
37    if ( num1 >= num2 ) {
38       printf( "%d is greater than or equal to %d\n", num1, num2 );
39    } /* end if */
40
41    return 0;   /* indicate that program ended successfully */
42
43 } /* end function main */
```

fig02_13.c (Part 2 of 2)

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

Program Output


```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 7 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

Program Output (continued)

Decisioni: Uguaglianza e Operatori Relazionali

Operatori				Associatività
*	/	%		Da sinistra a destra
+	-			Da sinistra a destra
<	<=	>	>=	Da sinistra a destra
==	!=			Da sinistra a destra
=				Da destra a sinistra

Fig. 2.14 Precedenza e associatività degli operatori discussi finora.

Parole chiave del linguaggio

Keywords			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Fig. 2.15 Parole chiave del linguaggio C.

Tipi di dati fondamentali

- **char**: un singolo byte, che rappresenta uno dei caratteri del set locale;
- **int**: un intero, il cui valore dipende dall'ampiezza degli interi sulla macchina utilizzata
- **float**: floating-point in singola precisione
- **double**: floating-point in doppia precisione
- Non esiste il tipo boolean (valori logici)
  - 0 rappresenta il valore logico FALSE
  - Un qualsiasi valore diverso da 0 rappresenta il valore logico TRUE (normalmente si usa !0)

Tipi di dati fondamentali

- In aggiunta i qualificatori
  - **short**
  - **long**
  - **signed**
  - **unsigned**
- Esempio
  - short int valore
  - long int prodotto
- I qualificatori **signed** e **unsigned** possono essere associati sia agli *int* che ai *char*
  - unsigned char assume valori tra 0 e 255
  - signed char assume valori tra -128 e 127

Tipi di dati fondamentali

- Il set di interi rappresentabili dipende dalla dimensione del tipo la quale non è ben definita
- Valgono le seguenti relazioni

1=sizeof(char)<sizeof(short)< sizeof(int)< sizeof(long)< sizeof(float)< sizeof(double)