Introduzione al linguaggio C

Laboratorio di
Linguaggi di Programmazione
a.a. 2001/2002

dott.ssa Francesca A. Lisi
lisi@di.uniba.it

Prerequisiti ed obiettivi

- Programmazione strutturata
- Teorema di Bohm-Jacopini
- Esperienza di programmazione in Pascal

- Operatività in tempi ridotti
- Approfondimenti *in itinere*

Testi consigliati

- “Linguaggi di programmazione”
  di R. Sethi
  edito da Zanichelli
- “Linguaggio C (ANSI C)” – II ed.
  di B.W. Kernighan, D.M. Ritchie
  edito da Jackson Libri
- "C - Corso completo di programmazione”
  di Harvey Deitel, Paul Deitel
  edito da Apogeo

Sommario (I parte)

- Richiami alla Programmazione Strutturata
- Caratteristiche di base del C
- Tipi primitivi
- Assegnamento
- Operatori aritmetici
- Operatori logici e relazionali
- Strutture di controllo
  - Sequenza
  - Selezione (if, if/else, switch)
  - Iterazione (while, do/while, for)

Richiami alla Programmazione Strutturata

- Rende i programmi più facili da comprendere, testare, correggere e modificare
- Utilizza solo strutture di controllo singolo-ingresso /singola-uscita
- Si basa su alcune **regole** pratiche:

  1) Comincia con il tracciare il “flowchart” più semplice

  2) Qualsiasi azione (blocco rettangolare) può essere rimpiazzato da due azioni in sequenza.

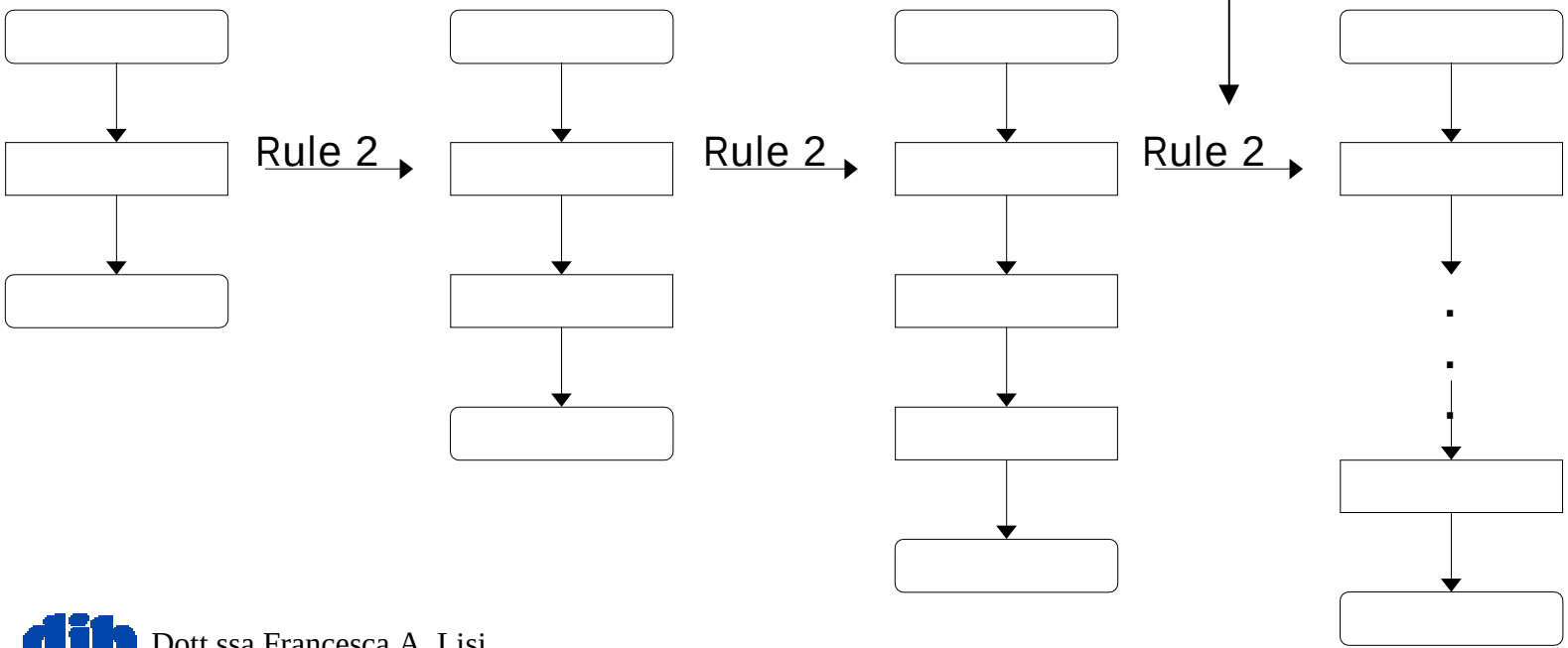
  3) Qualsiasi azione (blocco rettangolare) può essere rimpiazzato da una qualunque struttura di controllo (sequenza, selezione, iterazione).

  4) Le regole 2 e 3 possono essere applicate in qualsiasi ordine e in un numero svariato di volte.

Richiami alla Programmazione Strutturata (II)

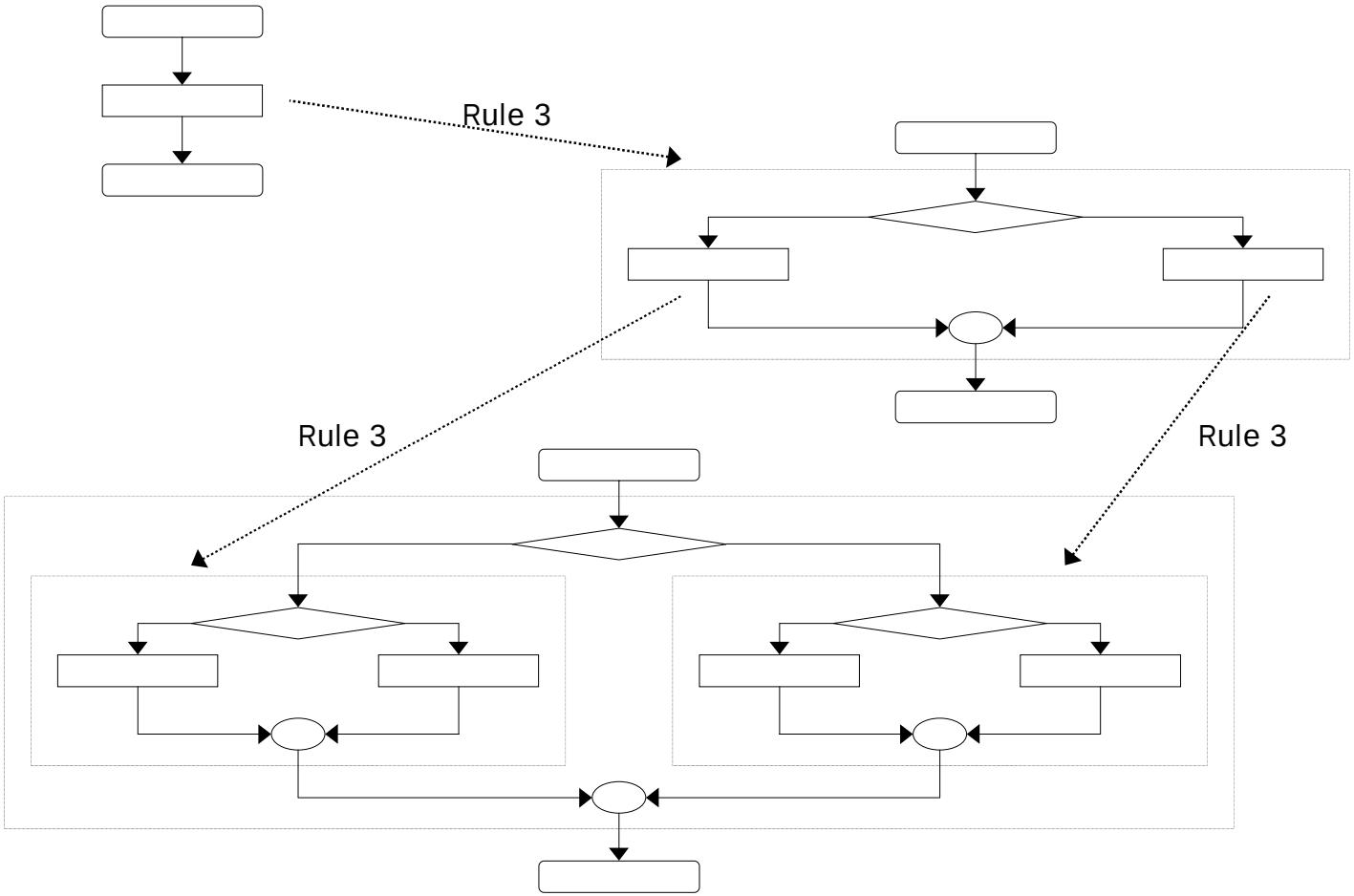
Rule 1 - Begin with the simplest flowchart

Rule 2 - Any rectangle can be replaced by two rectangles in sequence



Richiami alla Programmazione Strutturata (III)

Rule 3 - Replace any rectangle with a control structure



Caratteristiche di base del C

- Tendenza al basso livello
  - Corrispondenza con le istruzioni macchina
  - Efficienza
  - Sviluppato originariamente per UNIX, ora utilizzato per scrivere codice dei sistemi operativi
- Standard ANSI (1989-1999)
  - Definizione non ambigua
  - Indipendenza dalla macchina (portabilità)
- Tipizzazione debole
  - Controlli laschi
  - Comportamenti imprevisti

Un semplice programma C

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

```
Welcome to C!
```

Un semplice programma C (II)

- Testo delimitato da **`/*`** e **`*/`**
  - Racchiude commenti
  - Viene ignorato dal compilatore
- **`#include <stdio.h>`**
  - **`#include`** è una direttiva al pre-processore
  - **`<stdio.h>`** è un file di libreria che consente le operazioni I/O standard
  - Carica i contenuti del file di libreria

Un semplice programma C (III)

- **`int main()`**
  - I programmi C contengono una o più funzioni, di cui una esattamente deve essere **`main()`**
  - Le parentesi tonde **`()`** stanno ad indicare la parametricità della funzione
  - **`int`** significa che **`main()`** restituisce un valore intero
  - Le parentesi graffe **`{}`** racchiudono il corpo della funzione

- **`printf( "Welcome to C!\n" );`**
  - Chiede al calcolatore di stampare una stringa di caratteri fra virgolette e di andare a capo (**`\n`**)
  - L'intera riga di codice è detta "statement"
    - Tutti gli "statement" devono terminare con un punto e virgola

Tipi primitivi

Tipo	Tipo predefinito Pascal	Tipo predefinito C
numeri interi	`integer`	`int`
numeri reali a precisione singola	`real`	`float`
numeri reali a precisione doppia		`double`
caratteri	`char`	`char`
booleano	`Boolean`	`int (0=false)`

- Modificatori di tipo
  - **`signed`** e **`unsigned`** (si applicano a **`int`** e **`char`**)
  - **`short`** (si applica a **`int`**)
  - **`long`** (si applica a **`int`** e **`double`**)

Operatore di assegnamento =

Cfr Pascal `:=`

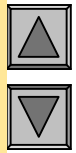
- **l-valori** (lvalues)
  - Espressioni che possono comparire sulla sinistra di un'equazione
  - I loro valori possono essere cambiati (es. **`x=4`** lecito)
- **r-valori** (rvalues)
  - Espressioni che possono solo apparire sulla destra di un'equazione
  - I loro valori sono costanti (es. **`4=x`** illecito)
- Gli l-valori possono essere usati come come r-valori, ma non viceversa (**`y=x`**)

Operatori aritmetici

Operazione	Operatore Pascal	Operatore C	Tipo operandi	Tipo risultato
somma	`+`	`+`	qualsiasi tipo aritmetico	stesso tipo dell'operando con tipo "superiore"
differenza	`-`	`-`		
prodotto	`*`	`*`		
divisione	`/`	`/`		
divisione fra interi	`div`		intero	intero
modulo	`mod`	`%`		

- Regole di precedenza
  - ()
  - `*`, `/`, `%` (se ce ne sono diversi, vanno valutati da sinistra verso destra)
  - `+`, `-` (se ce ne sono diversi, vanno valutati da sinistra verso destra)
- Regole di conversione
  - `long double`
  - `double`
  - `float`
  - `unsigned long int`
  - `long int`
  - `unsigned int`
  - `int`

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

int main()
{
   int integer1, integer2, sum;        /* declaration */

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

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



Outline

1. Initialize variables

2. Input

2.1 Sum

3. Print

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

Program Output

Un altro semplice programma C

- Come nel precedente esempio
  - Commenti, **`#include <stdio.h>`** e **`main`**
- **`int integer1, integer2, sum;`**
  - Dichiarazione di variabili (devono apparire prima delle istruzioni eseguibili)
  - **`int`** significa che le variabili possono assumere valori di tipo intero
  - **`integer1`**, **`integer2`**, e **`sum`** sono identificatori di variabili

Un altro semplice programma C (II)

- **`scanf( "%d", &integer1 );`**
  - Ottiene un valore dall'utente attraverso il canale di input standard (di solito, la tastiera)
  - Questa invocazione di **`scanf`** ha due argomenti:
    - **`%d`** - indica che i dati dovrebbero essere un intero decimale
    - **`&integer1`** - indirizzo di una locazione in memoria (variabile) denominata **`integer1`**
    - **`&`** va obbligatoriamente usato nelle chiamate di **`scanf`** (se ne parlerà più in là)
  - L'utente risponde a **`scanf`** digitando il numero, quindi battendo invio

Un altro semplice programma C (III)

- **`sum = integer1 + integer2;`**
  - **`sum`** è un l-valore
  - **`integer1 + integer2`** è un r-valore
  - Assegnamento del valore risultato dell'espressione **`integer1 + integer2`** alla variabile **`sum`**

- **`printf( "Sum is %d\n", sum );`**
  - Simile a **`scanf`**
  - **`%d`** significa che verrà stampato un intero decimale (in questo caso, il valore della variabile **`sum`**)

Un altro semplice programma C (IV)

- **`return 0;`**
  - Uno dei modi per uscire da una funzione
  - **`0`** significa che il programma è terminato normalmente
- **`}`**
  - Indica che la fine di **`main`** è stata raggiunta

Operatori logici e relazionali

Operazione	Operatore Pascal	Operatore C	Tipo operandi	Tipo risultato
prodotto logico	`and`	`&&`	booleano	booleano
somma logica	`or`	`||`	booleano	booleano
negazione logica	`not`	`!`	booleano	booleano

Operazione	Operatore Pascal	Operatore C	Tipo operandi	Tipo risultato
uguaglianza	`=`	`==`	qualsiasi tipo semplice	booleano
disuguaglianza	`<>`	`!=`		
minoranza stretta	`<`	`<`	tipo semplice o stringhe	
maggioranza stretta	`>`	`>`		
minoranza	`<=`	`<=`	tipo semplice	
maggioranza	`>=`	`>=`		

Lo zucchero sintattico del C

- Operatori di incremento e decremento `++` e `--`
  - *var* `++` equivale a *var* = *var*`+1`
  - *var* `--` equivale a *var* = *var*`-1`
  - `++` *var*, `--`*var*
- Operatori di assegnamento *op*=

  espr-1 op= *espr-2*

  equivale a

  espr-1 = (*espr-1*) op (*espr-2)*
  - `+, -, /, %, <<, >>, &, ^, |`
  - es. **`i += 2`**

La struttura di sequenza

Pascal

```
begin
   istruzione-1;
   istruzione-2;
   ...;
   istruzione-n
end
```

C

```
{
   istruzione-1;
   istruzione-2;
   ...;
   istruzione-n;
}
```

Le strutture di selezione `if` e `if/else`

Pascal

`if` *espressione*
`then` *istruzione*

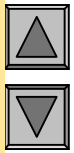
`if` *espressione*
`then` *istruzione1*
`else` *istruzione2*

C

`if (`*espressione***`)`**
istruzione

`if (`*espressione***`)`**
istruzione1
`else` *istruzione2*

```c
/* Fig. 2.13: fig02_13.c
   Using if statements, relational
   operators, and equality operators */
#include <stdio.h>

int main()
{
   int num1, num2;

   printf( "Enter two integers, and I will tell you\n" );
   printf( "the relationships they satisfy: " );
   scanf( "%d%d", &num1, &num2  );   /* read two integers */

   if ( num1 == num2 )
      printf( "%d is equal to %d\n", num1, num2 );

   if ( num1 != num2 )
      printf( "%d is not equal to %d\n", num1, num2 );

   if ( num1 < num2 )
      printf( "%d is less than %d\n", num1, num2 );

   if ( num1 > num2 )
      printf( "%d is greater than %d\n", num1, num2 );

   if ( num1 <= num2 )
      printf( "%d is less than or equal to %d\n",
              num1, num2 );
```



Outline

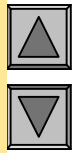
1. Declare variables

2. Input

2.1 `if` statements

3. Print

```
29
30    if ( num1 >= num2 )
31       printf( "%d is greater than or equal to %d\n",
32               num1, num2 );
33
34    return 0;   /* indicate program ended successfully */
35 }
```



Outline

3.1 Exit `main`

Program Output

```
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
```

```
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
```

La struttura di selezione multipla `switch`

Pascal

```
case espressione of
   cost-1: istruzione
   cost-2: istruzione
   ...
   cost-n: istruzione
end
```

C

```
switch (espressione)
{
   case cost-1: istruzione
   case cost-2: istruzione
   ...
   case cost-n: istruzione
   default: istruzione
}
```

Le strutture di iterazione `while` e `do/while`

Pascal

`while` *espressione* **`do`**
    istruzione`;`

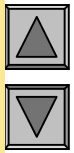
`repeat`
    istruzione
`until (`*espressione***`);`**

C

`while (`*espressione***`)`**
    istruzione

`do`
    istruzione
`while (`*espressione***`)`**

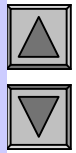
```c
/* Fig. 3.6: fig03_06.c
   Class average program with
   counter-controlled repetition */
#include <stdio.h>

int main()
{
   int counter, grade, total, average;

   /* initialization phase */
   total = 0;
   counter = 1;

   /* processing phase */
   while ( counter <= 10 ) {
      printf( "Enter grade: " );
      scanf( "%d", &grade );
      total = total + grade;
      counter = counter + 1;
   }

   /* termination phase */
   average = total / 10;
   printf( "Class average is %d\n", average );

   return 0;   /* indicate program ended successfully */
}
```



Outline

1. **Initialize Variables**
2. **Execute Loop**
3. **Output results**

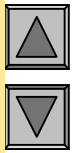
```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```



Outline

Program Output

```c
/* Fig. 3.10: fig03_10.c
   Analysis of examination results */
#include <stdio.h>

int main()
{
   /* initializing variables in declarations */
   int passes = 0, failures = 0, student = 1, result;

   /* process 10 students; counter-controlled loop */
   while ( student <= 10 ) {
      printf( "Enter result ( 1=pass,2=fail ): " );
      scanf( "%d", &result );

      if ( result == 1 )        /* if/else nested in while */
         passes = passes + 1;
      else
         failures = failures + 1;

      student = student + 1;
   }

   printf( "Passed %d\n", passes );
   printf( "Failed %d\n", failures );

   if ( passes > 8 )
      printf( "Raise tuition\n" );

   return 0;   /* successful termination */
}
```

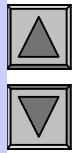


Outline

1. Initialize variables

2. Input data and count passes/failures

3. Print results

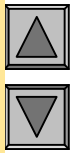


Outline

Program Output

```
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Passed 6
Failed 4
```

```c
/* Fig. 4.7: fig04_07.c
   Counting letter grades */
#include <stdio.h>

int main()
{
   int grade;
   int aCount = 0, bCount = 0, cCount = 0,
       dCount = 0, fCount = 0;

   printf(  "Enter the letter grades.\n"  );
   printf(  "Enter the EOF character to end input.\n"  );

   while ( ( grade = getchar() ) != EOF ) {

      switch ( grade ) {    /* switch nested in while */

         case 'A': case 'a':  /* grade was uppercase A */
            ++aCount;         /* or lowercase a */
            break;

         case 'B': case 'b':  /* grade was uppercase B */
            ++bCount;         /* or lowercase b */
            break;

         case 'C': case 'c':  /* grade was uppercase C */
            ++cCount;         /* or lowercase c */
            break;

         case 'D': case 'd':  /* grade was uppercase D */
            ++dCount;         /* or lowercase d */
            break;
```



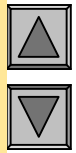
Outline

1. Initialize variables

2. Input data

2.1 Use switch loop to update count

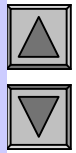
```
33
34          case 'F': case 'f':  /* grade was uppercase F */
35             ++fCount;         /* or lowercase f */
36             break;
37
38          case '\n': case' ':  /* ignore these in input */
39             break;
40
41          default:         /* catch all other characters */
42             printf( "Incorrect letter grade entered." );
43             printf( " Enter a new grade.\n" );
44             break;
45       }
46    }
47
48    printf( "\nTotals for each letter grade are:\n" );
49    printf( "A: %d\n", aCount );
50    printf( "B: %d\n", bCount );
51    printf( "C: %d\n", cCount );
52    printf( "D: %d\n", dCount );
53    printf( "F: %d\n", fCount );
54
55    return 0;
56 }
```



Outline

2.1 Use switch loop to update count

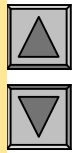
3. Print results



Outline

Program Output

```
Enter the letter grades.
Enter the EOF character to end input.
A
B
C
C
A
D
F
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
B

Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
```



Outline

1. Initialize variable

2. Loop

3. Print

```
1  /* Fig. 4.9: fig04_09.c
2     Using the do/while repetition structure */
3  #include <stdio.h>
4
5  int main()
6  {
7     int counter = 1;
8
9     do {
10       printf( "%d  ", counter );
11    } while ( ++counter <= 10 );
12
13    return 0;
14 }
```

```
1  2  3  4  5  6  7  8  9  10
```

Program Output

La struttura di iterazione `for`

Pascal

`for` *variabile* **`:=`** *espr2* **`to`** *espr3* **`do`**

istruzione

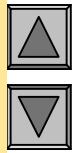
C

`for (`*espr1; espr2; espr3***`)`**

istruzione

```c
/* Fig. 4.5: fig04_05.c
   Summation with for */
#include <stdio.h>

int main()
{
   int sum = 0, number;

   for ( number = 2; number <= 100; number += 2 )
      sum += number;

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

   return 0;
}
```

```
Sum is 2550
```



Outline

4.6 Examples Using the for Structure

Program to sum the even numbers from 2 to 100

Program Output

La prossima volta ...

- Struttura di un programma C
- Funzioni
- I/O base
- Puntatori
- Tipi derivati (vettori, strutture)
- Introduzione all'ambiente Borland C++