Fondamenti di Informatica
Modulo B
A.A. 2003/2004

- Docente: Prof. Alessandro Sperduti
- E-mail: sperduti@math.unipd.it
- Tel: 049-8275982
- Ricevimento: giovedì dalle 15 alle 17
  (Dip. Matematica Pura ed Applicata, via Belzoni 7)

- 16 ore di lezione
  - Dicembre: 10, 11, 17, 18
  - Gennaio: 7, 8, 14, 15
- Pagina del corso:
  - http://www.math.unipd.it/~sperduti/fondamentiB.html

Sommario degli argomenti

Concetti di Base

Linguaggi di Programmazione

- Linguaggi per impartire istruzioni al processore
- Programma
  - sequenza di istruzioni
  - normalmente pensato per risolvere un problema di calcolo
  - al programma vengono forniti dei dati
  - il programma calcola eseguendo le istruzioni
  - il programma restituisce i risultati

Linguaggi di programmazione

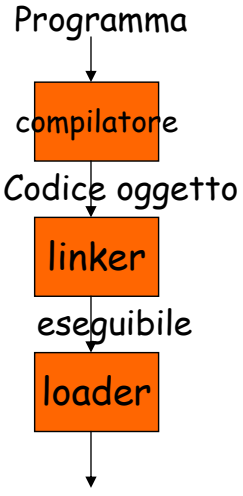
- Linguaggi ad alto livello (rispetto all'Assembler)
- Linguaggi imperativi: rispecchiano le operazioni reali nell'architettura di Von Neumann
  - Scrivere un valore in una cella di memoria o in un registro
- Es.: Fortran (Formula Translator), 1954
- Algol, Cobol, APL, LISP ('60), Pascal, Prolog ('71), C ('74), Ada ('79), C++ ('85), Java ('94)

Elementi dei linguaggi di programmazione

- Alfabeto: alfabeto inglese, cifre decimali (0,1,…,9), punteggiatura (;), simboli di operazioni (+,-,*,…)
- Parole chiave: es. IF, THEN, WHILE, …
- Operatori: aritmetici, logici (and, not, or, …), relazionali (<, >, =, …)
- Tipi di dati: interi, reali, caratteri, …
- Sintassi: regole per scrivere un programma
- Semantica: significato dell'esecuzione di un programma

Da linguaggio X a linguaggio macchina

- Se X= Assembler ➔ assemblatore
- Se X piu' ad alto livello ➔ compilatore o interprete
- Fasi di un compilatore:
  - Analisi lessicale: da caratteri a nomi, operatori, parole chiave, … (tokens)
  - Analisi sintattica/semantica: da tokens a costrutti (if then else, while, …)
- Generazione codice oggetto in linguaggio macchina
- Linker/loader: collegamento tra vari programmi e scelta degli indirizzi di RAM



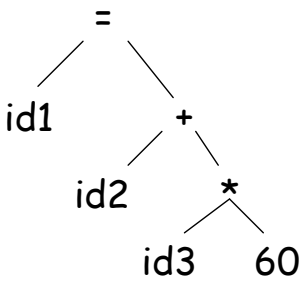
Programma in ling. macchina in RAM

Esempio

- p = p + r * 60 ← sequenza di caratteri
- Analisi lessicale:

id1 = id2 + id3 * 60 ← sequenza di tokens

- Analisi sintattica/semantica
- Generazione codice intermedio:
  - temp1 = 60
  - temp2 = id3*temp1
  - temp3 = id2 + temp2
  - id1 = temp3
- Ottimizzazione codice:
  - temp1 = id3 * 60
  - id1 = id2 + temp1
- Generazione codice oggetto:
  - LOAD R2 id3
  - LOAD R3 60
  - MULT R2 R3
  - LOAD R1 id2
  - ADD R1 R2
  - STORE R1 id1



Struttura di un programma C++

```
#include <iostream.h>
<altre eventuali direttive>

void main ()
{
   <dichiarazioni>

   <operazioni>

}
<funzione1>
<funzione2>
 …
```

Un primo esempio di programma

```
// il mio primo programma in C++

   #include <iostream.h>

   int main ()
   {
     cout << "Salve gente!";
     return 0;
   }
```

```
Salve gente!
```

Nella parte sinistra è riportato il codice del nostro primo programma che noi chiameremo, ad esempio, salve.cpp.

La parte destra mostra quello che si ottiene eseguendolo.

Comando di compilazione in Linux: **`g++ -o salve salve.cpp`**

- Richiama il compilatore g++ per compilare il programma scritto nel file salve.cpp
- Il risultato viene scritto nel file eseguibile **`salve`**

Un primo esempio di programma

```
// il mio primo programma in C++

   #include <iostream.h>

   int main ()
   {
     cout << "Salve gente!";
     return 0;
   }
```

Riga di commento. Tutte le righe che iniziano con due sbarrette (//) vengono considerate commento e non hanno alcun effetto sul comportamento del programma.

Direttiva. Le frasi che iniziano con il simbolo di cancelletto (#) sono direttive per il preprocessore del compilatore.
In questo caso la direttiva richiede di includere il file della libreria standard **`iostream.h`** che contiene le dichiarazioni delle operazioni basilari di input-output definite nella libreria standard del C++

La funzione **main** è il punto da cui inizia l'esecuzione di un qualsiasi programma C++. **main** è seguito da una coppia di parentesi **( )** perché esso è una funzione. In C++ tutte le funzioni sono seguite da una coppia di parentesi **( )** che possono contenere degli argomenti. La dichiarazione (l'intestazione) della funzione è seguita dal contenuto (il corpo) della funzione racchiuso tra parentesi graffe (**{ }**).

Un primo esempio di programma

```
// il mio primo programma in C++

   #include <iostream.h>

   int main ()
   {
     cout << "Salve gente!";
     return 0;
   }
```

cout è il flusso standard di output del C++ (di solito indirizzato allo schermo), e l'effetto della istruzione **<<** è quello di inserire una sequenza di caratteri (nel nostro caso "Salve gente!") in tale flusso di output.

La dichiarazione di **cout** si trova nel file **iostream.h**, ed è per questo che è stato incluso tale file.

La frase finisce con un punto e virgola ";", che indica la fine dell'istruzione e deve essere messo alla fine di ogni istruzione (uno degli errori più comuni è appunto dimenticare il punto e virgola alla fine di una istruzione.

L'istruzione **return** fa terminare la funzione **main()** e ritorna quello che è indicato di seguito, nel nostro caso **0**.
Poiché il valore ritornato è un intero bisogna dichiarare **main** come una funzione che ritorna un intero (**int main ()**).
Molti compilatori inseriscono automaticamente una istruzione **return** alla fine di una funzione.

Un primo esempio di programma

Il corpo del programma è stato diviso in più righe per renderlo più leggibile, ma si può scrivere lo stesso programma in un'unica riga:

```
int main () { cout << "Salve gente!"; return 0;}
```

- In C++ la separazione tra istruzioni è indicata dal punto e virgola che termina ciascuna di esse. La suddivisione del programma in più righe serve a rendere il programma più leggibile alle persone che lo leggono, per il compilatore la cosa non fa' alcuna differenza.
- Questa regola non vale per le direttive per il preprocessore (quelle che iniziano con **#**) in quanto esse sono istruzioni per il preprocessore che verranno lette ed interpretate dal preprocessore e che spariranno dal codice che verrà compilato dal compilatore.
- Esse devono comparire ciascuna in una propria riga e non richiedono il punto e virgola alla fine.

Un secondo esempio di programma

Ecco un programma con qualche altra istruzione:

```
// il mio secondo programma in C++

   #include <iostream.h>

   int main ()
   {
     cout << "Salve gente!";
     cout << “Sono un programma C++”;
     return 0;
   }
```

```
Salve gente! Sono un programma C++
```

Il cui corpo può essere anche scritto in una sola riga:

```
int main () { cout << "Salve gente!"; cout << “Sono un programma C++”; return 0;}
```


Commenti

I commenti sono parti del testo che vengono ignorate dal compilatore.
In C++ ci sono due modi di inserire commenti:

```
// riga di commento
/* blocco di commento */
```

Il primo considera commento tutto ciò che segue la coppia di sbarre (**//**) fino alla fine della riga. Il secondo considera commento tutto ciò che è compreso tra la coppia di caratteri **/*** e la prima occorrenza successiva della coppia di caratteri ***/**, **eventualmente anche più righe di testo**.

```
/* il mio secondo programma in C++
   con più commenti */

   #include <iostream.h>

   int main ()
   {
     cout << "Salve gente!"; // dice Salve gente!
     cout << “Sono un programma C++”;  // dice Sono un programma C++
     return 0;
   }
```

Variabili

- Variabile: nome associato ad una cella di memoria
- Può assumere un valore (quello contenuto nella cella)
- Istruzione di assegnamento per dare un valore ad una variabile (es: `a = 5`)
- Valore ottenuto calcolando un'espressione (es: `a + 1`)

Es.

```
a = 5;
b = 2;
a = a + 1;
risultato = a - b;
```

Ogni variabile necessita di un identificatore (un nome) che la distingue da ogni altra variabile.
Ad esempio nel codice precedente gli identificatori sono **`a`** , **`b`** e **`risultato`**, ma avremmo potuto chiamare le variabili con un qualsiasi altro nome inventato da noi, purchè esso sia un identificatore valido.

Identificatori

- Un identificatore valido è una sequenza di una o più lettere, cifre o simboli di sottolineatura (_).
- La lunghezza di un identificatore non è limitata, ma alcuni compilatori considerano significativi soltanto i primi 32 caratteri .
- Un identificatore non deve contenere spazi o altri caratteri. Sono permessi soltanto lettere, cifre e simboli di sottolineatura. Inoltre, gli identificatori di variabile devono sempre iniziare con una lettera.
- In nessun caso un identificatore può iniziare con una cifra.
- Un'altra regola da tener presente quando si scelgono gli identificatori è che essi non devono essere uguali ad una delle **parole chiave del linguaggio** nè ad una di quelle specifiche del compilatore usato.

Tipi di Dato

- I valori di una variabile (un carattere, un piccolo numero, un grande numero,…) possono essere di vario tipo e quindi possono occupare una quantità di memoria diversa nel calcolatore
- La memoria di un calcolatore è suddivisa in byte. Il byte è la più piccola quantità di memoria che possiamo gestire, esso può contenere una relativamente piccola quantità di dati: di norma un singolo carattere o un intero fra 0 e 255.
- Il calcolatore può elaborare anche tipi di dato più complessi che occupano più di un byte.
- Vediamo di seguito un elenco dei tipi fondamentali del C++ con l'indicazione della quantità di memoria necessaria e del rango di valori che si possono rappresentare:

Tipi di Dato

Nome	Byte*	Descrizione	Rango*
char	1	carattere o intero di 8 bit.	**signed:** -128 ... 127 **unsigned:** 0 ... 255
short	2	intero di 16 bit.	**signed:** -32768 ... 32767 **unsigned:** 0 ... 65535
long	4	intero di 32 bit.	**signed:**-2147483648 ... 2147483647 **unsigned:** 0 ... 4294967295
int	*	Intero. La sua lunghezza dipende dalla lunghezza del tipo **word** usato dal sistema operativo. Ad esempio, in MSDOS è di 16 bit mentre in sistemi a 32 bit (quali Windows 9x/2000/NT) è di 32 bit (4 bytes).	Vedi **short**, **long**
float	4	numero in virgola mobile.	3.4e + / - 38 (7 cifre decimali)
double	8	numero in virgola mobile in doppia precisione.	1.7e + / - 308 (15 cifre decimali)
long double	10	numero in virgola mobile in doppia precisione estesa.	1.2e + / - 4932 (19 cifre decimali)
bool	1	Valori Booleani. Può assumere uno dei due valori: true o false. NOTA: è un tipo aggiunto recentemente allo standard ANSI-C++. Non tutti i compilatori lo accettano.	**true** or **false**
wchar_t	2	Carattere esteso. Viene usato per rappresentare tutti i caratteri internazionali. NOTA: è un tipo introdotto recentemente nello standard ANSI-C++. Non tutti i compilatori lo accettano.	caratteri estesi

* I valori nelle colonne Byte e Range possono variare a seconda del sistema.

Dichiarazioni delle Variabili

- Prima di usare una variabile occorre dichiararla specificando a quale tipo di dato essa appartenga.
- La sintassi di una dichiarazione di variabile prevede prima il nome del tipo di dato (quale **int**, **short**, **float**, ...) seguito dall'identificatore scelto per denotare tale variabile. Ad esempio:

```
int a;
float
numero;
```

- Una volta dichiarate, le variabili **a** e **numero** possono essere usate nel programma all'interno del loro campo di validità (lo scope).
- Se vogliamo dichiarare più di una variabile dello stesso tipo possiamo farlo in una stessa riga indicando una sola volta il tipo e separando gli identificatori con la virgola. Ad esempio:

```
int a, b, c;
```


Dichiarazioni delle Variabili

- I tipi di dato interi (**char**, **short** , **long** e **int**) possono essere signed o unsigned a seconda del rango di numeri che si vuole considerare.
- Per specificare un intero possiamo usare una delle parole chiave **signed** o **unsigned** prima del nome del tipo. Ad esempio:

```
unsigned short NumeroDiFigli;
signed int IlMioSaldoBancario;
```

- Se non si specifica né **signed** né **unsigned** viene assunto **signed**
- Possiamo usare **signed** e **unsigned** come nomi di tipo con lo stesso significato di **signed int** e **unsigned int** rispettivamente. Le seguenti due dichiarazioni sono equivalenti:

```
unsigned AnnoDiNascita;
unsigned int AnnoDiNascita;
```

Un altro esempio di programma

```
// operazioni con le variabili

   #include <iostream.h>

   int main ()
   {
     // dichiarazione delle variabili:
     int a, b;
     int risultato;

     // elaborazione:
     a = 5;
     b = 2;
     a = a + 1;
     risultato = a - b;

     // stampa del risultato:
     cout << risultato;

     // terminazione del programma:
     return 0;
   }
```

```
4
```


Inizializzazione delle Variabili

- Quando dichiariamo una variabile il suo valore è indeterminato
  (i bit della zona di memoria riservata alla variabile hanno il valore che era stato loro assegnato da qualche precedente programma).
- Se nel dichiarare una variabile vogliamo anche assegnarle un valore iniziale, basta aggiungere alla dichiarazione un simbolo di uguale seguito dal valore desiderato:

```
tipo identificatore = valore_iniziale ;
```
← Stile C

```
tipo identificatore(valore_iniziale);
```
← Stile C++

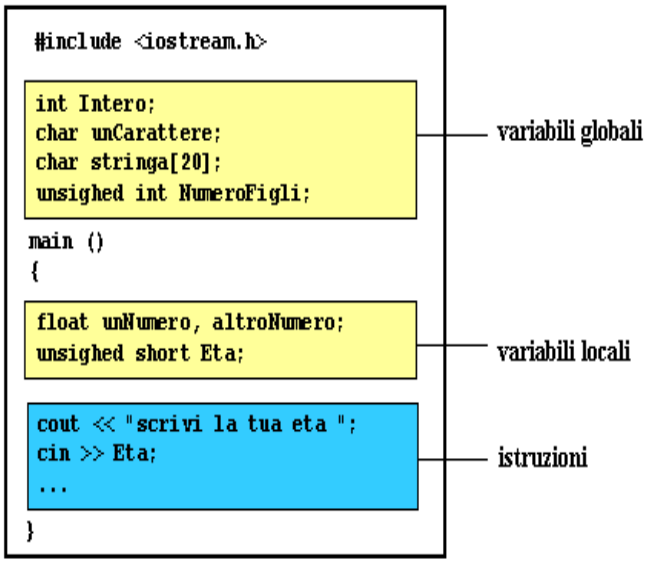
- Ad esempio se vogliamo dichiarare una variabile a di tipo **int** con valore iniziale 0 possiamo scrivere:

```
int a = 0;
```
← Stile C

```
int a(0);
```
← Stile C++

In C++ si può utilizzare sia lo stile C che lo stile C++

Scope delle Variabili



Le **variabili globali** si possono usare in tutto il programma dal punto in cui sono dichiarate fino alla fine.

Lo scopo delle **variabili locali** è invece limitato al blocco in cui sono dichiarate. Se sono dichiarate all'inizio di una funzione (come in **main**) il loro scopo è l'intero corpo della funzione. Se nell'esempio ci fosse un'altra funzione diversa da **main()**, le variabili locali dichiarate in **main** non sarebbero visibili all'interno di tale funzione e viceversa.

In C++ lo scopo di una variabile locale è limitato alla parte del blocco in cui esse sono dichiarate che segue la dichiarazione stessa (un blocco è un gruppo di istruzioni racchiuse tra parentesi graffe {}.

Costanti

- Una costante è una qualsiasi espressione che ha un valore prefissato.
- Esse si possono suddividere in Numeri Interi, Numeri in Virgola Mobile, Caratteri e Stringhe.
- Per scrivere una costante intera non occorre usare le virgolette (") o qualche altro carattere speciale:

```
75    // decimale
0113  // 75 in ottale (occorre premettere il carattere 0)
0x4b  // 75 in esadecimale (occorre premettere i due caratteri 0x)
```

- Numeri in virgola mobile (numeri con una parte frazionaria e/o un fattore esponenziale):

```
3.14159     // 3.14159
6.02e23     // 6.02 x 10^23
1.6e-19     // 1.6 x 10^-19
3.0         // 3.0
```

Costanti

- Caratteri e Stringhe:

```
'z'
'p'
"Salve gente"
"Come state?"
```

caratteri

stringhe

- I letterali che rappresentano singoli caratteri sono racchiusi tra due caratteri apice (') mentre i letterali che rappresentano stringhe sono racchiusi tra due caratteri doppio apice (").
- Questo è necessario per poter distinguere valori di tipo stringa da valori di tipo carattere (che sono considerati diversi).
- Inoltre, l'uso degli apici (') e (") evita di confondere un letterale carattere o stringa da un identificatore o una parola chiave:
  x denota la variabile x, mentre `'x'` denota la costante di tipo carattere 'x'

Costanti

- Per rappresentare con un letterale (o all'interno di un letterale stringa) alcuni caratteri speciale si usano notazioni particolari (i codici di escape).
- Ecco una lista di tali codici di escape (ognuno di essi inizia con il carattere barra rovesciata (\):

\n	a capo riga	**\r**	ritorno carrello	**\t**	tabulazione
\v	tabulazione verticale	**\b**	backspace	**\f**	nuova pagina
\a	allerta (beep)	**\'**	apice singolo (')	**\"**	doppio apice (")
\?	punto interrogativo (?)	****	barra rovesciata (\ )		

Esempio:

```
'\n'
'\t'
"Sinistra \t Destra"
"uno\ndue\ntre"
```

Costanti Definite (#define)

- Possiamo usare la direttiva **#define** del preprocessore per dare un nome ad una costante nel seguente modo:

```
#define identificatore_di_costante
```

- Ovunque useremo l'identificatore, il preprocessore provvederà a sostituirlo con la costante.

```
#define PI 3.14159265
#define NEWLINE '\n'
#define WIDTH 100
```

Esempio di uso:

```
circonferenza = 2 * PI * r;
cout << NEWLINE;
```

- **#define** non è una istruzione C++ ma una direttiva per il preprocessore: deve quindi essere scritta in una sua propria riga e non deve essere aggiunto il carattere ; alla fine.
- Una volta che abbiamo associato un identificatore ad una costante possiamo usare tale identificatore in ogni punto seguente del programma come se esso fosse una costante.

Costanti Dichiarate (const)

- Usando il prefisso **const** si possono dichiarare delle costanti appartenenti ad un determinato tipo esattamente allo stesso modo in cui si dichiarano le variabili:

```
const int larghezza = 100;
const char tab = '\t';
const int cap =  12440;
```

- In realtà le costanti dichiarate sono semplicemente delle variabili il cui valore non può più essere modificato
- Siccome una volta create non è più possibile cambiarne il valore, esse devono essere sempre inizializzate con un valore al momento della loro creazione.