Inf Gen II

INFORMATICA GENERALE II

Ingegneria delle Telecomunicazioni
Università di Trento

Marco Roveri
roveri@irst.itc.it

Introduzione al C++ (I)

(Materiale preso ed adattato da materiale del Prof. M. Benedetti)

AA 2006/2007

Inf Gen II

C++

- C
  - Evoluzione da due linguaggi pre-esistenti: BCPL e B effettuata da Ritchie.
  - Utilizzato per sviluppare UNIX
  - Utilizzato per sviluppare sistemi operativi moderni (Linux, Windows, Mac OS X, …).
  - Hardware independent
  - Standard creato nel 1989 e modificato nel 1999.
- C++
  - Sovrainsieme del C sviluppato da Stroustrup ai Bell Labs.
  - Fornisce Object Oriented capabilities.
  - Ora linguaggio dominante in accademia e industria (anche se Java sta prendendo piede).

AA 2006/2007

Inf Gen II

C++ vs Java vs JavaScript

- C++ e Java
  - Linguaggi *compilati*.
  - Linguaggi di programmazione ad oggetti: *completi* e *complessi*.
  - Adatto a progetti di *grandi dimensioni*.
  - *Fully extensible*: programmatori possono creare i propri oggetti e strutture dati.
  - *Strong typing*: i tipi delle variabili devono essere dichiarati e non possono cambiare.
  - Java è "*platform independent*", C++ è *platform dependent*.
- JavaScript
  - Linguaggio *interpretato* "facile da usare".
  - *Limitata* programmazione ad oggetti.
  - *Loose typing*.
  - Adatto a progetti di *piccole* dimensioni.
  - Forte integrazione con pagine HTML.
  - JavaScript come Java è "*platform independent*".

Inf Gen II

Perchè C++

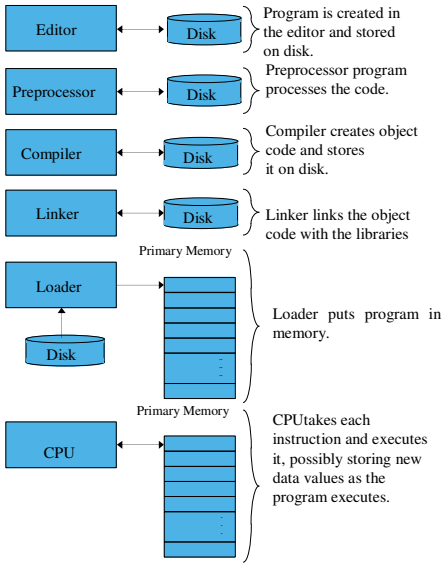
- È fondamentale conoscere più di un linguaggio di programmazione.
- È largamente utilizzato nell'industria e in accademia per la realizzazione di progetti complessi.
- È un linguaggio multi-paradigma (imperativo, object-oriented).
- C++ è un superset del C, più ricco e di più alto livello.
- Il C++ standard comprende una ampia libreria (una volta chiamata Standard Template Library) che fornisce al programmatore una ampia gamma di oggetti generici predefiniti (alcune di queste le vedremo nell'ambito di questo corso).

Programma C++

- Programma C++ è una sequenza *finita* di istruzioni che:
  - Risolvono un problema.
  - Soddisfano i seguenti criteri:
    - Ricevono valori in ingresso.
    - Producono valori in uscita.
    - Sono chiare, non ambigue ed eseguibili.
    - Terminano dopo un numero finito di passi.
  - Operano su strutture dati.

C++:

- Fasi dello sviluppo di un programma C++:
  - Edit
  - Preprocess
  - Compilation
  - Link
  - Load
  - Execute



Inf Gen II

C++: Lessico

- C++ è *case sensitive.*
- *Blanks*, *Tabs*, *Newline* sono ignorati tranne quando elementi di una stringa.
- *Blanks* possono essere aggiunti per aumentare la leggibilità del codice.
- I commenti del codice come in JavaScript:
  - // solo per i commenti in linea.
  - /* … */ per commenti che possono finire su più linee.
- Keywords, sono parole riservate del linguaggio di programmazione.
- Identificativi sono: nomi di variabili, funzioni, procedure, oggetti.
  - Il primo carattere deve una lettera, un underscore.
  - I caratteri successivi possono essere una lettera, un numero o un underscore.
  - Un identificatore *deve* essere diverso da una qualunque delle keyword.
  - Identificatore: [A-Za-z_][A-Za-z0-9_]*

Inf Gen II

C++: keywords

break bool case catch char class const continue default delete do double else enum export extern false float for friend goto if inline int long main namespace new NULL operator private protect public return short signed sizeof static std struct switch template this throw true try typeof union using virtual void volatile while

C++: costanti

- Sequenze di caratteri che denotano un particolare valore che non cambia durante l'esecuzione.
- Esempi:
  - Boolean: *true* e *false*
  - Numerica: 5, 2.4, 0xFF, 3E10, -4.5E-9
  - Stringhe: *"marco"*, *"\tstudenti telecomunicazioni\n"*.
  - Primitive: *NULL*

C++: operatori

- Alcuni caratteri speciali e le loro combinazioni sono usati come *operatori*.

```
+    -    *    /    %    ^    &    |    ~
!    =    <    >    +=   -=   *=   /=   %=
^=   &=   |=   <<   >>   >>=  <<=  ==   !=
<=   >=   &&   ||   ++   --   ,    ->*  ->
.*   ::   ()   []   ?:
```

C++: variabili e costanti

- Le *variabili* e le *costanti* sono utilizzate per memorizzare un *valore* in una area di memoria.
- Le variabili consentono la *modifica* del loro valore durante l'esecuzione del programma.
- L'area di memoria corrispondente è identificata da un *nome*, che ne individua l'*indirizzo di memoria.*
- Le variabili e le costanti sono quindi caratterizzate da: *nome* (l'identificatore), *tipo*, *locazione di memoria* (*left-value*) e *valore* (*right-value*).

left-value	right-value
nome	4E10

C++: definizione vs dichiarazione

- **Definizione**: quando il compilatore incontra una definizione di una variabile, esso predispone l'allocazione di un'area di memoria in grado di contenere la variabile del tipo scelto.
  - Sintassi: *tipo identificatore [= expressione];*
  - *Esempio:*
    - ***int*** *x;*
    - ***int*** *x = 3 * 2*
- **Dichiarazione**: specifica solo il tipo della variabile e presuppone dunque che la variabile venga definita in una altra parte del programma.
  - Sintassi: ***extern*** *tipo identificatore;*
  - *Esempio:*
    - ***extern int*** *x;*

C++: dichiarazione di costanti

- Sintassi:

  const *tipo identificatore = expressione;*

  Deve essere possibile calcolare il valore dell'espressione staticamente in fase di compilazione.

- Esempi:

  const int kilo = 1024;

  const double pi = 3.141519;

  const int mille = kilo – 24;

C++: Tipi fondamentali e derivati

- In C++ si distinguono:
  - **Tipi fondamentali**: servono a rappresentare informazioni semplici, come i numeri interi, i caratteri (**bool, int**, **short, long, char, enum, float, double**)
  - **Tipi derivati**: permettono di costruire strutture dati complesse a partire dai tipi fondamentali/derivati (puntatori, riferimenti, array, strutture, union, classi)

C++: operatori

- Manipolano e comparano variabili, costanti, valori.
  - Aritmetici: + - * / % ++ --
  - Assegnazione: = += -= *= /= %=
  - Confronto: == != > >= < <=
  - Logico: ! && ||
  - Bit: & | ^ ~ - << >>
  - Condizione: ?

C++: operatore di assegnazione

- La sintassi dell'operatore di *assegnazione semplice*
  exp1 = exp2
- ***exp1*** deve essere un'espressione dotata di l-value
- ***exp1*** e ***exp2*** devono essere di tipo compatibile
- Il valore denotato da un'espressione di assegnazione è il valore di exp2.
- Un'assegnazione può essere usata all'interno di un'altra espressione.
- L'operazione di assegnazione, '**=**', associa a destra.
- Esempio:
  - **int** a, b, c, d;

    a = b = c = d = 5;
  - equivalente a:

    (a = (b = (c = (d = 5))));

C++: operatori di assegnazione

Forma compatta	Forma Estesa
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

C++: operatori di incremento

x++	*incrementa* **x** e denota il valore di **x** *prima* dell'incremento
x--	*decrementa* **x** e denota il valore di **x** *prima* del decremento
++x	*incrementa* **x** e denota il valore di **x** *dopo* l'incremento
--x	*decrementa* **x** e denota il valore di **x** *dopo* il decremento

Inf Gen II

AA 2006/2007

©MR

C++: espressioni e istruzioni

- Le **espressioni** sono costrutti che descrivono come calcolare un particolare valore
  - 3 + 2
  - (4 * 3) + 5 == 6
- Gli **statement** o **istruzioni** sono azioni che il processore può elaborare.
  - In C++ uno statement è composto da una o più espressioni.
  - Gli statement sono delimitati dal carattere "**;**"
  - Esempio:
    - y = x + 1;
    - z = (!x || y) && (x || !y);

19

Inf Gen II

AA 2006/2007

©MR

C++: espressioni

- Valgono le stesse considerazioni viste per JavaScript.
- Le espressioni vengono valutate left to right seguendo l'albero sintattico indotto dalla priorità degli operatori e dalle parentesi.
- Il valore di una espressione può essere calcolato senza valutare l'intero albero sintattico (**lazy-evaluation**)

a = 0, b = 0;

(a < b) && (((a + b) + 1) == 0) ? a : b; → false

20

C++: statement condizionali

- Statement condizionale **if-then**

```
if ( <condizione> ) {
    // Blocco istruzioni se la <condizione> è vera
}
```

- Statement condizionale **if-then-else**

```
if ( <condizione> ) {
    // Blocco istruzioni se la <condizione> è vera
}
else {
    // Blocco istruzioni se la <condizione> è false
}
```


C++: statement condizionali

- Statement condizionale **if-then-else** innestati

```
// Attenzione alle parentesi: i due frammenti di codice sotto sono diversi
if ( <condizione1> ) {
   // blocco istruzioni se <condizione1>  è vera
   if ( <condizione2> ) {
         // Blocco istruzioni se la <condizione2> è vera
   }
   else {
        // Blocco istruzioni se la <condizione2> è false
   }
}

if ( <condizione1> ) {
   // blocco istruzioni se <condizione1>  è vera
   if ( <condizione2> ) {
        // Blocco istruzioni se la <condizione2> è vera
   }
}
else {
    // Blocco istruzioni se la <condizione1> è false
}
```

C++: statement condizionali

- **switch** statement

```
switch ( <integer_expression> ) {
  case v1 : { /* blocco istruzioni */ } break;
  case v2 : { /* blocco istruzioni */ } break;
  ....
  default: { /* blocco istruzioni */ } break;
}
```


C++: ripetizione controllata

- **while** statement

```
while ( <cond> ) {
  /* blocco istruzioni da ripetere */
}
```

- **for** statement

```
for ( <init stm>; <cond>; <inc stm> ) {
  /* blocco istruzioni da ripetere */
}
```

- **do-while** statement

```
do {
  /* blocco istruzioni da ripetere */
} while ( <cond> );
```

C++: array

- Motivazione
  - Supponiamo di dover memorizzare i coefficienti di un polinomio di grado *n*:

$$P(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

  - Dobbiamo dichiarare tante variabili quanti sono i coefficienti – ognuna con un nome diverso – e poi usarle opportunamente.

    int a0, a1, …., an;

  - Se da un polinomio di grado *n* passiamo a manipolare un polinomio di grado *2n* dobbiamo aggiungere *n* nuove variabili distinte.
  - Inoltre se il grado del polinomio non è noto a priori in fase di compilazione del programma, che cosa posso fare?
  - In matematica, per affrontare situazioni simili, si sceglie un unico nome per il parametro/coefficiente e poi si ricorre alla notazione *con pedice*: si usa la scrittura "a_i" per indicare l'i-esimo coefficiente di un insieme di coefficienti $\{a_i, i=0,\ldots,n\}$.

C++: array

- In C++ esiste una possibilità del tutto analoga (come in JavaScript):
- Invece di scrivere a_i (il pedice non si può indicare in un file sorgente, essendo un documento di puro testo) in C++ si scrive: **a[i]**
- L'oggetto **a** nell'insieme prende il nome di **array**; un array è quindi un raccoglitore di tante variabili, che hanno un nome formato da una parte iniziale uguale al nome dell'array, e si distinguono poi per il loro *indice* indicato tra parentesi quadre.
- Come ogni altro oggetto C++, prima di poterlo usare un array lo si deve *definire*; ad esempio:

  int a[100]

  dichiara che utilizzeremo un array di nome "a", che contiene 100 variabili di tipo **int;**
- Quando si definisce un array come "**int a[100]**", il compilatore riserva in memoria uno spazio di memoria sufficiente a memorizzare (in maniera adiacente) 100 variabili di tipo intero.
- Come in JavaScript, gli indici con cui si accederà a tali variabili vanno da **0** a **99**.
- In generale se definisco un array di **n** elementi esso corrisponderà a variabili il cui indice è compreso tra **0** e **n-1**.
  - Nota bene: l'indice di un array **NON** è compreso tra **1** e **n**.

C++: array

- I singoli elementi di un array di interi sono - a tutti gli effetti - variabili di tipo intero. Lo stesso vale ovviamente per array di qualunque altro tipo (di **boolean**, **float**, **char**,…). Si può dunque:
  - Assegnare un valore ad una *cella* dell'array:

    ```
    a[3] = 10;
    ```
  - Coinvolgere elementi dell'array in espressioni:

    ```
    a[6] = (a[8] = a[3]*a[3]+2)+1;
    ```
  - Utilizzare variabili (intere) o espressioni per *indicizzare* l'array:

    ```
    int i=1, b=5;
    a[i--] = ++b;
    a[i] = 5*a[i+3]+b;
    a[++b] = ++a[a[i+1]];
    ```
  - **Non** si può invece operare con assegnazioni tra array;
    - se a e b sono due array di interi, scrivere "a=b;" è un errore.

C++: array statici e dinamici

- Quando si scrive, ad esempio:

  float coef[20];

  è già noto a tempo di compilazione lo spazio necessario a memorizzare l'array: si tratta di 20 volte quello necessario a memorizzare un singolo **float**;
- Il compilatore può dunque predisporre lo spazio necessario alla memorizzazione dell'array così come farebbe con ogni altra variabile dichiarata;
- Tuttavia, il C++ - a differenza di altri linguaggi - consente anche la definizione *dinamica* degli array senza variazioni nella sintassi d'utilizzo;

C++: array statici e dinamici

- Nelle definizioni dinamiche la dimensione dell'array è un valore calcolabile solo a tempo d'esecuzione; ad esempio, si può scrivere:

  char c[len];

  per definire un array c di caratteri di capacità "len", dove len è una variabile o più in generale un'espressione.
- Questo aggiunge notevole flessibilità al linguaggio; la gestione delle complicazioni aggiuntive (l'array deve trovare spazio in memoria a tempo di esecuzione e non di compilazione, spazio di una dimensione anch'essa nota solo *durante* l'esecuzione) viene delegata al C++: non se ne occupa il programmatore.
- Vedremo più avanti maggiori dettagli sugli array dinamici.

C++: array - inizializzazione

- Come nel caso della definizione di una variabile semplice, anche per gli array si può specificare un valore iniziale per gli elementi dell'array.

  int a[10] = {5, 2, -5, 10, 234, 0, 1, 100, -1, 3};

C++: array di caratteri e stringhe

- Un array di caratteri può essere usato per rappresentare stringhe, e in questo caso esiste il vincolo che l'**ultima posizione dell'array non può rappresentare un carattere significativo, ma deve contenere il valore speciale zero**;
- Tale valore serve a varie funzioni che manipolano stringhe a capire *dove finisce la stringa*. Dunque, il numero di caratteri necessari per memorizzare una stringa lunga **n** caratteri è **n+1**;
- Esiste una forma semplificata di inizializzazione utilizzabile per usare un array di caratteri come stringa. La sua sintassi è (esempio):

  char mia_stringa[] = "Ciao a tutti!";
- Vengono riservati in memoria 14 spazi, e la situazione risultante è:

C	i	a	o		a		t	u	t	t	i	!	
67	105	97	111	32	97	32	116	117	116	116	105	33	0

- Si può stampare questa stringa sullo schermo con il semplice comando:

  cout << mia_stringa;

C++: array di caratteri e stringhe

- Esistono molte funzioni di libreria per manipolare array di caratteri che rappresentano stringhe: funzioni per calcolare la lunghezza di una stringa, per concatenare stringhe, per verificare la presenza di una certa sottostringa all'interno di un'altra stringa, e molte altre ancora. Le vedremo più avanti;
- Questo modo di gestire le stringhe è ereditato dal C, ed ha pregi e difetti. Uno dei difetti principali consiste nella "difficoltà" di gestire dinamicamente la lunghezza delle stringhe. Ad esempio, per concatenare due stringhe ad ottenerne una terza è necessario gestire esplicitamente il calcolo e l'allocazione del nuovo array necessario a contenere il risultato;
- Per ovviare a simili problemi il C++ mette a disposizione un meccanismo più flessibile, la classe "String" (che **non** studieremo in questo corso), che permette di maneggiare le stringhe in modo molto naturale e del tutto simile al modo utilizzato in JavaScript.

C++: array multidimensionali

- Come in JavaScript, in C++ si possono definire array bidimensionali (multidimensionali):

  int a[10][5];

  Corrisponde alla definizione di un array di due dimensioni – matrice – con 10 righe e 5 colonne.
- Gli indici con cui si accede alla matrice vanno, come al solito, da 0 alla dimensione meno uno (da zero a 9 per le righe, da zero a 4 per le colonne, nel caso dell'esempio precedente); per accedere all'elemento in posizione di riga i e di colonna j si usa la sintassi: a[i][j];
- Il numero di variabili intere allocate dalla definizione di un array del tipo

  int a[N][M];

  è pari ad NxM, e il valore iniziale di ogni elemento è, al solito, **indefinito**;
- È possibile definire ed usare array a tre o più dimensioni, utilizzando una coppia aggiuntiva di parentesi quadre per ogni dimensione; ad esempio:

  float a[10][5][20];

C++: array multidimensionali

- Gli array bidimensionali possono essere inizializzati in fase di definizione con una scrittura di questo tipo (esempio):

```
int a[4][5] = { { 2, 5, -8, 7, 6  },
                { 3,10,  7, 6, 1 },
                {-1, 8, -8, 5, 3  },
                { 2, 5,  8, 4, 2  } };
```

- Anche se noi accediamo e manipoliamo gli elementi di un array a due dimensioni come se esso fosse a tutti gli effetti una rappresentazione bidimensionale, è chiaro che la sua rappresentazione in memoria è comunque *lineare*; in particolare, il C++ memorizza in memoria le matrici una riga dopo l'altra. Ad esempio, l'array a precedente si trova memorizzato in memoria come qui sotto illustrato:

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{30}	a_{31}	a_{32}	a_{33}	a_{34}
2	5	-8	7	6	3	10	7	6	1	-1	8	-8	5	3	2	5	8	4	2

- Vedremo più avanti ulteriori considerazioni sugli array multidimensionali e le relazioni con i puntatori.