Introduzione C++

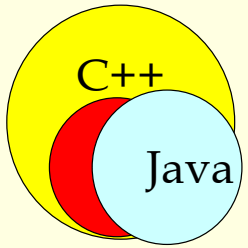
Compilatori

Librerie

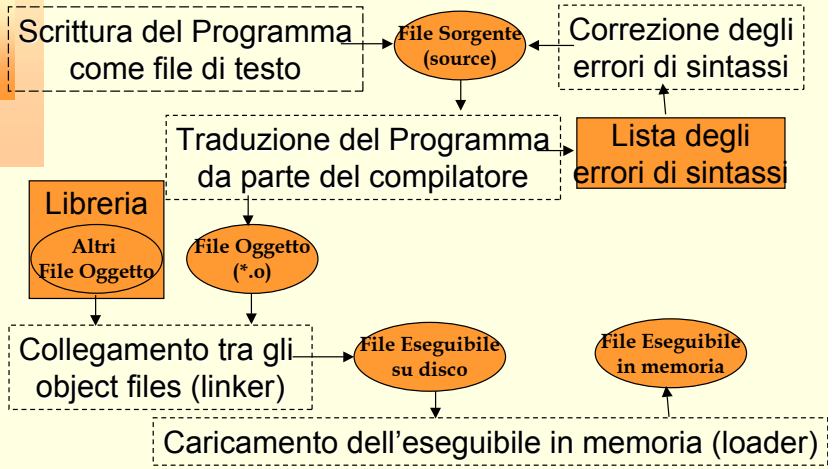
Usare oggetti

Introduzione a C, C++ e Java

- 1977 C
- 1986 C++
  - "a better C" con estensioni agli oggetti
  - oggi uno standard industriale
- 1994 Java
  - "C ++ --"

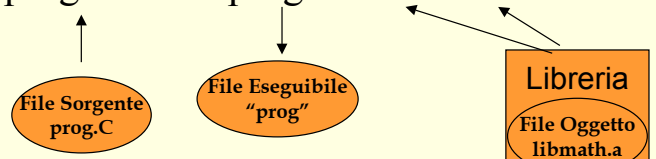


Il compilatore

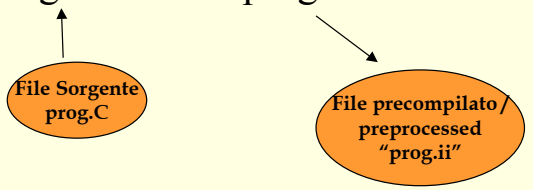


La compilazione in g++

- g++ prog.CPP -o prog -Ldir -lmath

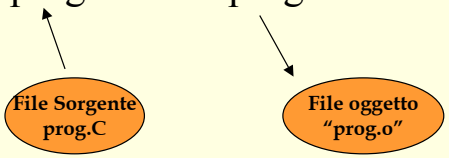


- g++ prog.CPP –E > prog.ii

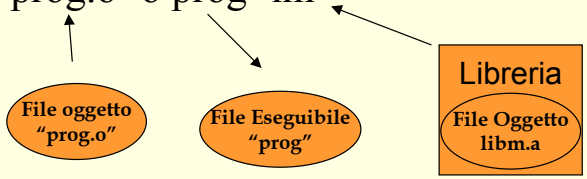


La compilazione in g++

- g++ prog.CPP –c prog.o

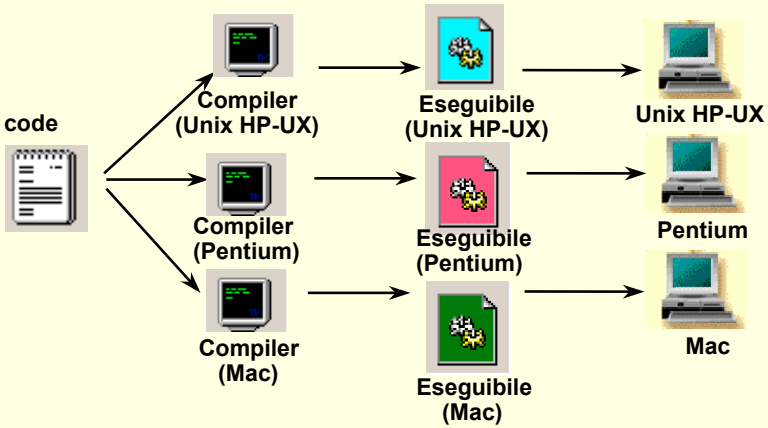


- ld prog.o -o prog -lm



Portabilità

- Programmi tradizionali (C o C++)



Type Checking

- Statico (compile-time): il compilatore controlla la correttezza degli argomenti passati alle funzioni
  - implementato in C++
- Dinamico (run-time): il compilatore inserisce codice per effettuare il controllo durante l’esecuzione
  - implementato in Java
  - *utile, ma crea “overhead”*

Dichiarazioni

- dichiarazione: si introduce un nome/identificatore al compilatore (variabile o funzione)

```
int fun1 ( int , int );
```

dichiarazione: la funzione fun1: prende due interi e restituisce un intero

```
extern int i;
```

dichiarazione: la variabile i è di tipo intero

Definizioni

- definizione: crea spazio/codice per l'oggetto (variabile o funzione)

```
int fun1 ( int lenght, int width)
{ int area = lenght*width;
  return area; }


int i;
```


Esempi

```
extern int i;            // dichiarazione

int f1 (int x, int y); // dichiarazione

int j;   // dichiarazione e definizione

int f2 (int x) {         // dich. e def.
     return x*x;
}
```


Librerie e Headers

- le librerie contengono un grande numero di variabili e funzioni
- per supportarne e controllarne l'uso il C++ impiega degli "header files" (.h)
  - contenitori delle dichiarazioni delle variabili e delle funzioni usate nella libreria
  - il programmatore che crea la libreria deve fornire il file di header
- **il file .h è la dichiarazione dell'interfaccia**

Include format

- notazione in C

```
# include <caratteri.h>
# include "caratteri.h"
```

Nel *path* di ricerca

Nella *directory* locale

- notazione in C++

```
# include <mia_libreria>
# include "mia_libreria"
```

per le librerie standard del C

```
# include <stdio.h>
   oppure
# include <cstdio>
```

Namespaces

- Problema del C: alla fine si “finiscono” i “nomi” per variabili e funzioni
- C++ offre un meccanismo per ovviare al problema
  - Ogni set di definizioni in una libreria è “confezionato” *wrapped* in un “spazio di nomi” *namespace*
  - Definizioni con lo stesso nome ma in *namespaces* diversi non creano collisioni

Uso di Namespaces

- Indicazione esplicita:

```
# include <iostream>
using namespace std
```

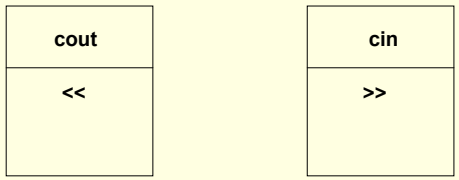
- Dichiarazioni equivalenti (backward compatibility):

```
# include <iostream.h>

# include <iostream>
using namespace std
```


Uso delle classi <iostream>

```
#include <iostream>

using namespace std;
```

- permette nei programmi l’utilizzo delle classi e dei metodi della libreria standard del C++ per l’input/output da files (compresi stdin e stdout)



L’oggetto cout

```
#include <iostream>
using namespace std;

int main() {
  cout << "Hello, World! I am "
       << 8 << " Today!" << endl;
}
```

Operatore “output”

stringa

Variabile definita in iostream per newline (\n)

Overloading / manipolatori

```
// More streams features
#include <iostream>
using namespace std;
int main() {
  // Specifying formats with manipulators:
  cout << "a number in decimal: "
       << dec << 15 << endl;
  cout << "in octal: " << oct << 15 << endl;
  cout << "in hex: " << hex << 15 << endl;
  cout << "a floating-point number: "
       << 3.14159 << endl;
  cout << "non-printing char (escape): "
       << char(27) << endl;
}
```


L'oggetto cin

```
// Converts decimal to octal and hex
#include <iostream>
using namespace std;

int main() {
  int number;
  cout << "Enter a decimal number: ";
  cin >> number;
  cout << "value in octal = 0"
       << oct << number << endl;
  cout << "value in hex = 0x"
       << hex << number << endl;
}
```


La classe <string>

- Le *stringhe* in C++ come in C sono **array di caratteri** (tipo char) terminati dal carattere speciale '/0'
- La classe **string** è usata in C++ per nascondere all'utente le manipolazioni a basso livello
  - *chiedere ai programmatori di C !!*

Standard C++ string class

```
#include <string>
#include <iostream>
using namespace std;
int main() {
  string s1, s2; // Empty strings
  string s3 = "Hello, World."; // Initialized
  string s4("I am"); // Also initialized
  s2 = "Today"; // Assigning to a string
  s1 = s3 + " " + s4; // Combining strings
  s1 += " 8 "; // Appending to a string
  cout << s1 + s2 + "!" << endl; }
```

La classe <fstream>

- La libreria <fstream> fornisce al programmatore un modo semplice per lavorare con i file
- In molte implementazioni <fstream> include <iostream>
  - per sicurezza provare con il vostro compilatore

Copia di un file su un altro file

```
#include <string>
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  ifstream in("Scopy.cpp");   // Open for reading
  ofstream out("Scopy2.cpp"); // Open for writing
  string s;
  while(getline(in, s))  // Discards newline char
    out << s << "\n";    // ... must add it back
}
```


Copia di un file su una stringa

```
#include <string>
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  ifstream in("FillString.cpp");
  string s, line;
  while(getline(in, line))
    s += line + "\n";
  cout << s;
}
```


<vector> contenitori generici

- A differenza di un array, non è necessario predeterminarne la dimensione
- E' un *template* -- classe parametrica, si tratta di una classe che può essere utilizzata con più tipi di dato.
- Uso del template <vector>
  - *vector<string> v* // per la dichiarazione
  - *v.push_back(arg)* // metodo per inserire elementi
  - *v[12]* // metodo per estrarre elementi

Copia di file su vettore di stringhe

```
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

int main() {
  vector<string> v;
  ifstream in("Fillvector.cpp");
  string line;
  while(getline(in, line))
    v.push_back(line); // Add the line to the end
  // Add line numbers:
  for(int i = 0; i < v.size(); i++)
    cout << i << ": " << v[i] << endl;
}
```


Separazione delle parole

```
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

int main() {
  vector<string> words;
  ifstream in("GetWords.cpp");
  string word;
  while(in >> word)
    words.push_back(word);
  for(int i = 0; i < words.size(); i++)
    cout << words[i] << endl;
}
```


I vettori anche con i numeri

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> v;
  for(int i = 0; i < 10; i++)
    v.push_back(i);
  for(int i = 0; i < v.size(); i++)
    cout << v[i] << ", ";
  cout << endl;
  for(int i = 0; i < v.size(); i++)
    v[i] = v[i] * 10; // Assignment
  for(int i = 0; i < v.size(); i++)
    cout << v[i] << ", ";
  cout << endl;
}
```


esercizi

- Modificare FillVector in modo che stampi in ordine inverso le linee del file in input
- Modificare FillVector in modo che attenda un input dall'utente prima di stampare una linea del file in input