Laboratorio Trattamento Numerico Dati Sperimentali

Fernando Palombo

3^0 Piano Edificio LITA

e-mail: palombo@mi.infn.it

URL: http://idefix.mi.infn.it/~palombo

Organizzazione del Corso

- Il corso consiste di 24 ore frontali tenute da me in questa aula ogni lunedi e di 36 ore di laboratorio che si terranno di pomeriggio (mercoledi', giovedi' o venerdi' a seconda del proprio turno) tenute dal Prof. A. Andreazza, L. Carminati e D. Maino

- Suddivisione turni:

Orario delle lezioni ore 8:30 - 10:30 (in pratica..)

- Le lezioni frontali saranno di supporto alle applicazioni pratiche di laboratorio

- L'esame finale consiste in una prova **scritta** e in una prova **orale**

Obiettivi del Corso

- Insegnarvi a programmare (anche ad oggetti) in C++

- Le applicazioni si baseranno essenzialmente su problemi di calcolo numerico in Fisica (ricerca di zeri e minimi di una funzione, interpolazione, integrazione, tecniche Monte Carlo, equazioni differenziali ordinarie)

- Fondamentale seguire lezioni ed esercitazioni e poi:

  PRATICA, PRATICA, PRATICA, ….

- Perche' C++ e perche' questo corso: …..

- Il programma di questo anno accademico e' un po' diverso da quello degli anni passati. Le prime lezioni saranno un richiamo (e approfondimenti su OOP) di argomenti trattati nel corso di Informatica al I^{o} anno)

Materiale Didattico

- La bibliografia di testi di C++ e' molto vasta. Molti tutorial sono disponibili gratis in rete.

- Documentazione su C++ si puo' trovare in: http://www.cplusplus.com/

- Un ottimo tutorial e' quello di J. Soulie :

  http://www.cplusplus.com/doc/tutorial

  Questo tutorial in formato pdf puo' essere scaricato (gratis). In buona parte seguiro' questo tutorial.

- Comunque qualunque altro testo puo' andare bene lo stesso.
  Un testo di C++ ormai classico e' quello di Bjarne Stroustrup (l'architetto di C++) : Il linguaggio C++ (anche in edizione italiana). Questo pero' e' un testo avanzato !

Materiale Didattico

- Le lezioni (trasparenze) e le applicazioni pratiche delle lezioni si trovano in;
  http://idefix.mi.infn.it/~palombo/didattica/Lab-TNDS/CorsoLab
  Ad ogni lezione sono associate le corispondenti applicazioni.

- Attualmente trovate anche :

tutorialC++.pdf	Tutorial di C++
Getting-Started.pdf	Tutorial UNIX di SLAC. Ottimo, completo eavanzato.
unix.htm	UNIX for BaBar , ottimo (ma in alcuni punti specifico di BaBar ed avanzato).
unixtut	Ottimo e completo Unix tutorial

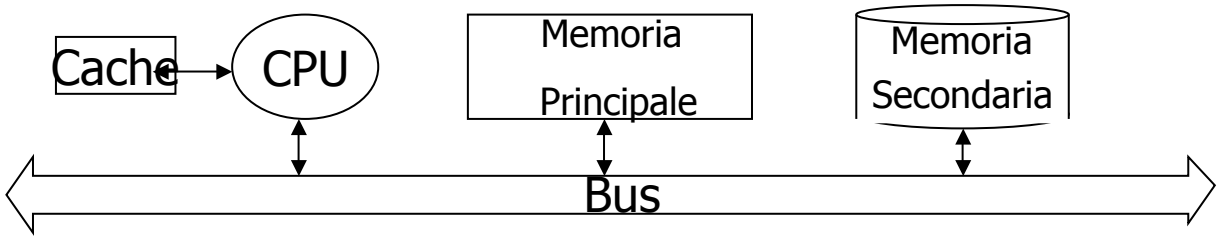
- Tutto il materiale e' scaricabile sul vostro PC. I tutorial funzionano anche standalone

- Sito del laboratorio :

  http://labmaster.mi.infn.it/Laboratorio2/labTNDS

Hardware del Computer

Componenti hardware del computer



- Central Processing Unit (CPU): il cuore del computer dove vengono eseguite le varie operazioni aritmetiche e logiche.

- Random Access Memory (RAM) : e' la zona di memoria principale. E' volatile l'informazione memorizzata .
  Switch on (1) e off(0). Ogni switch e' chiamato bit. 8 bit formano un byte

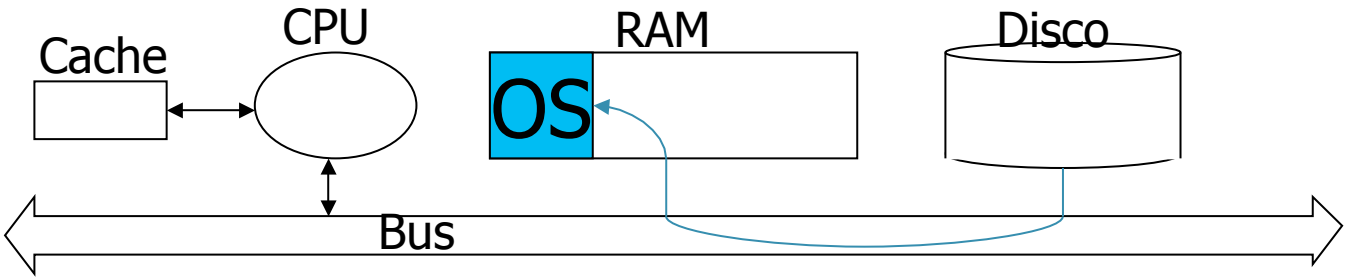
- Memoria Secondaria (hard disK, CD-ROM, ecc). Lettura piu' lenta. Lunga memorizzazione e grande spazio disponibile.

- BUS: connette la CPU a tutti gli altri dispositivi.

- Cache: accedere alla RAM e' abbastanza veloce ma non quanto la velocita' di esecuzione nella CPU. Per questo motivo si associa alla CPU una memoria detta cache che memorizza le istruzioni e dati recentemente usati.

Software del Computer

Componenti software del computer



- Il sistema operativo (OS) risiede nella memoria secondaria. Quando il computer viene accesso OS viene caricato nella memoria centrale dove rimane sino a che il computer viene spento.

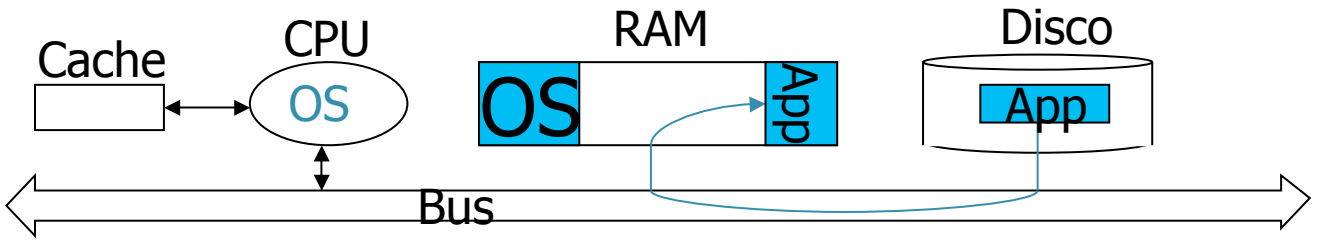
- OS: controlla il funzionamento e interazione dei vari componenti hardware del computer. Gestisce l'interazione dell'utente col computer -> UNIX, Linux, MacOS, Windows-XP, ecc

- Applicazioni (App): programmi NON-OS che svolgono una qualche utilita' , tipo word processor, browser, compilatori , ecc

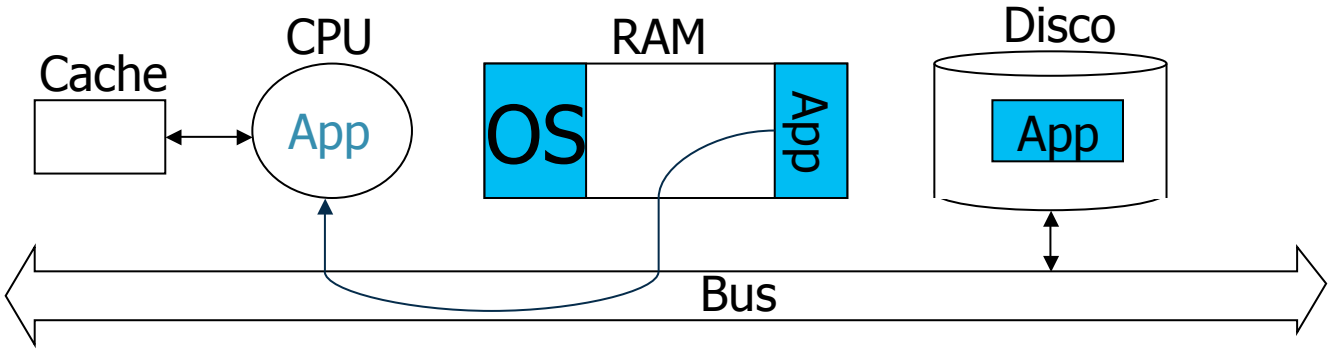
- Programmi Utente: sono i programmi che scrive l'utente per risolvere un qualche problema: risoluzione numerica di equazioni differenziali, analisi statistica di un campione di dati sperimentali, ordinare una successione di numeri, disegnare qualcosa, ecc, ecc

Software del Computer

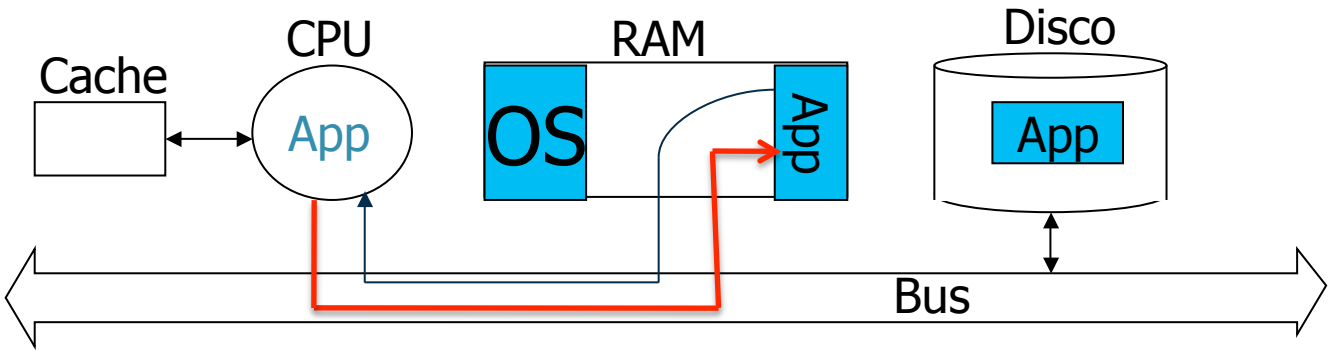
- Quando si lancia un programma, OS controlla la CPU e carica il programma dal disco nella RAM



- A questo punto OS lascia la CPU al programma che comincia a runnare



Software del Computer



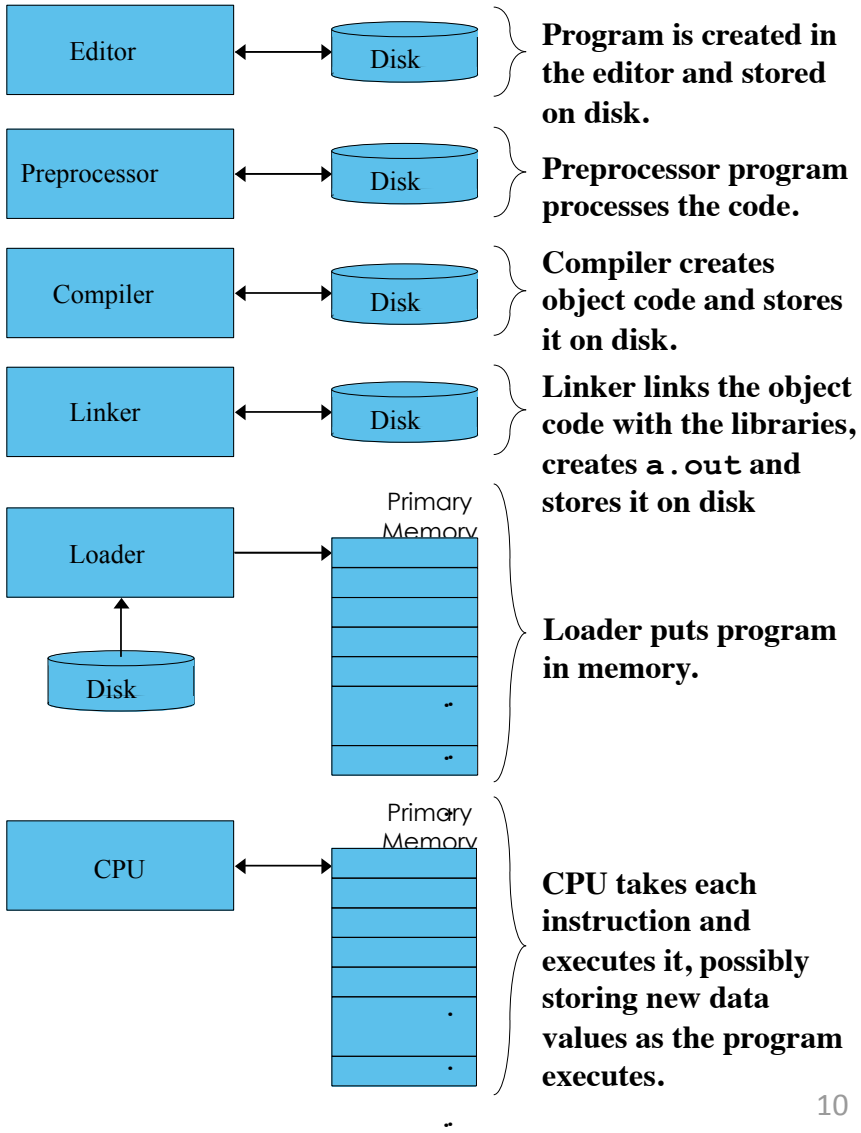
- Quando il programma entra in esecuzione ripetutamente va a prendere la successiva istruzione dalla memoria/cache, la esegue e memorizza qualunque risultato ottenuto nella memoria.

- Questo processo di prendere una istruzione , eseguirla e mettere il risultato indietro nella memoria viene ripetuto milioni di volte al secondo.

- Il collo di bottiglia e' nel bus.

Fasi di un Programma C++

- Scrittura sorgente programma
- Preprocessing
- Compilazione
- Link
- Loader
- Esecuzione



Basic Input/Output

- La libreria di input e output standard di C++ permette all'utente di stampare messaggi (usualmente) sullo schermo e di ottenere dati in input dalla tastiera.

- C++ usa speciali oggetti (stream) per assolvere a questi compiti. Gli oggetti stream di input (cin), di output (cout) e stream degli errore (cerr) sono dichiarati nello header file iostream

- Standard Output (cout). Lo standard output di default e' lo schermo. Si accede ad esso tramite cout utilizzando l'operatore inserzione <<

```
cout << "ciao" ;       // scrive ciao sullo schermo
cout << " x =" << 120 <<endl;  // scrive x=120 sullo schermo;
endl;  termina correttamente la riga e passa a quella successiva
```

- Standard input (cin) : di default e' la tastiera. Si applica l'operatore estrazione (>>) sulla stream cin : int age; cin >> age;
  Il sistema aspetta che da cin (tastiera) venga dato un input che verra' assegnato alla variabile age (di tipo int)
- Standard error (cerr) : la stream cerr mostra i messaggi di errore

Input/Output con File

- Le classi che regolano la lettura e scrittura di caratteri da file o in file sono Istream e ostream. Da queste si derivano le classi ofstream (che permette la scrittura su un file) , ifstream (che permette la lettura da file) e fstream (che permette entrambe le cose)

- fstream myfile;
  myfile.open("example.txt", ios::out) ; //apri per scrivere . Ci sono diversi flag
  myfile << "Ciao" << endl;
  myfile.close(); // chiudo il file

- open() e' un metodo che ha due argomenti: il nome del file ed il modo in cui si apre il file (ios::in, ios::out, ios::app,, ios:: binary, ecc.) . Il modo puo' avere piu' di un valore separati dall'operatore | . Per esempio
  myfile.open("example.txt", ios::out | ios:: app);
  qui voglio scrivere ma appendendo a quanto preesistente (non voglio sovrascrivere su altri dati che ho nel file!)

Input/Output con File

```
// writing on a text file
#include <iostream>
#include <fstream>
using namespace std;
int main () {
ofstream myfile ("example.txt"); // apro in scrittura nel modo default
if (myfile.is_open())  //verifico che myfile sia stato effettivamente aperto
{
myfile << "This is a line” << endl;
myfile << "This is another line” << endl;
myfile.close();  // chiudo il file myfile
}
else cout << "Unable to open file";
return 0;
}
```

Input/Output con File

- Il controllo dell'apertura del file si puo anche fare cosi;

```
fstream    myfile; // su questo file si puo' leggere e scrivere
if(!myfile){  //ritorna falso se il file non e' stato aperto
cout << "non posso aprire il file" << endl;
return 1;}
```

- Si puo' controllare quando si giunge alla fine del file utilizzando il metodo eof() [che ha prototipo int eof();]

- Questo metodo ritorna non zero quando viene raggiunta la fine del file e zero negli altri casi

- Esistono molti altri metodi che permettono di operare con le stream in input e output. Qualcuno lo vedremo in seguito.

Visibilita’ e Tempo di vita

- Scope o ambito di visibilita’ e’ l’insieme di istruzioni racchiusa da due parentesi graffe { …..}. Una variabile definita in uno scope e’ locale e riconosciuta in quello scope ma non e’ riconosciuta in uno scope diverso. Una variabile puo’ essere definita (e con lo stesso nome) in scope diversi.

- Se una variabile e’ definita fuori di ogni scope e’ detta variabile globale e lo scope di questa variabile e’ detto globale: una variabile globale e’ visibile in tutto il programma.

- In presenza di una variabile definita localmente in uno scope e anche globalmente , allora viene utilizzata solo la variabile locale. Se si vuole utilizzare la variabile globale si specifica la variabile col prefisso :: (detto operatore di risoluzione dello scope) .
  Questo operatore dice al compilatore che deve cercare fuori dello scope corrente una variabile con lo stesso nome

Il codice delle esemplificazioni pratiche che faremo in questa lezione lo trovate in : http://idefix.mi.infn.it/~palombo/didattica/Lab-TNDS/CorsoLab-2010/Applicazioni-Web/Intro1/

Scope: globale e locale

```
#include <iostream>
#include <iomanip>
using std::cout;
using std::cin;
using std::endl;
using std::setprecision;  //serve <iomanip>
using std::setw;   //serve <iomanip>
//using namespace std;

int square ( int ); // prototipo della funzione square (di un intero con ritorno di un int
double square ( double ); // protot. della funz. square (di un double con ritorno di int int
// esempio di overloading del nome di funzione che vedremo tra poco
a=2;  //var globale
//double a =2.6; //var globale
double b=3.234; //var globale
int M=55;
 int main() {
int M =6; //var locale; e' solo nello scope della funzione
cout << " M locale al main  = " << M << endl;
cout << " M globale = " << ::M << endl; // :: Scope Resolution Operator
cout << "a =" << a << ", Quadrato di a =" << square(a) << endl;
cout << "b =" << b << ", Quadrato di b =" << square(b) << endl;
```

```
cout << "b =" << b << ", Quadrato di b =" << setprecision(5) << square(b) << endl;
 cout << "b =" << b << ", Quadrato (int) di b =" << int(square(b)) << endl;
// cout << "quadrato di a = " << l << endl; // Errore l non dichiarata in questo scope
return 0;
}
int  square (int y) {  int y= y*y;  //var locale
cout <<"Sono in square; l = " << l << endl;
return y;    //la funzione fa una copia di y che riporta alla funzione chiamante}

double square (double y){  double l;  l=y*y;  return y*y;}
==========================================================
g++ -o scope scope.cc
./scope //eseguo il programma  (eseguibile in questa directory)

 M locale al main  = 6
M globale = 55
Sono in square; y = 4
a =2, Quadrato di a =4
a =2a non e' stato cambiato da square
b =3.234, Quadrato di b =10.4588
b =3.234, Quadrato di b =10.459         (con setprecision(5)
b =3.234, Quadrato (int) di b =10
```

Variabile Statica

```
/*calcola la media corrente di numeri dati dallo user.  Questo codice e'
nel file StaticVariable.cc*/

#include <iostream>
using namespace std;
double MediaCorrente (double x);  //prototipo della funzione

int main() {
  //leggi e fai la media (esci appena leggi -1)

  double numero;
  do   {
    cout << "entra un numero  (-1 per uscire): " << endl;
    cin >> numero;
    if(numero  != -1) cout<<  "   La media corrente e' :" << MediaCorrente (numero) << endl;
  } while (numero  != -1);
  return 0;
}
```

```
double  MediaCorrente (double num) {
  static double somma=0, contatore=0;
  //  double somma=0, contatore=0;
   // provare a togliere static e vedere  l'effetto che fa
  somma = somma +num;
  contatore++;
  return somma/contatore;
}

=========================================
 g++ -o StaticVariable StaticVariable.cc

./StaticVariable

entra un numero  (-1 per uscire):
10
   La media corrente e' :10
entra un numero  (-1 per uscire):
20
   La media corrente e' :15
entra un numero  (-1 per uscire):
-1
```

Area di Memoria Stack

- Questa e' la zona di memoria dove viene allocato automaticamente il pacchetto di dati quando l'esecuzione passa da una funzione (chiamante) ad un'altra funzione (chiamata). Questo pacchetto contiene l'indirizzo di rientro nella funzione chiamante, la lista degli argomenti passati alla funzione chiamata e tutte le variabili (locali) definite nella funzione chiamata.

- Questo pacchetto viene impilato sull'analogo pacchetto della funzione chiamante (LIFO)

- Questo pacchetto e' automaticamente eliminato dalla memoria quando l'esecuzione della funzione termina e si rientra nella funzione chiamante. Tutte le informazioni contenute in questo pacchetto sono perse.

- Eventuali variabili globali o variabili statiche sono allocate in una diversa zona di memoria che ha tempo di vita (lifetime) dell'intero programma. Queste variabili sono comunque visibili solo all'interno del loro scope

Passaggio di Parametri nelle Funzioni

```
//esempio di funzioni con parametri passati da copie (by value)

#include <iostream>
using namespace std;

int addition (int x, int y); // possibile anche int addition (int, int);
int main () {

  int z, x=3, y=5;
cout << " x = " << x << ", y = " << y <<  endl;
  z= addition (x,y);// copie di x e y sono passate alla funzione addition
  cout << " x = " << x << ", y = " << y << " and  the sum is " << z << endl;
}

int addition (int a, int b) {
  int r;
  r = a + b;
  //voglio vedere cosa succede se scambio in questo scope  a con b
```

Passaggio di Parametri nelle Funzioni

```
 int temp =a;
  a = b; // sto scambiando le copie delle variabili x=3 e y=5 della funzione main
  b = temp; //queste copie pero' sono nella memoria stack e scompaiono quando si
                  //torna nel main
  return (r);
  /*viene fatta una copia di r. Questa copia e' assegnata come valore di z nel
programma  chiamante */
}
```

==========

```
g++ -o FunzioneByValue FunzioneByValue.cc
./FunzioneByValue

 x = 3, y = 5 and  the sum is 8
```

I valori di x e y nel main non sono stati cambiati!

Passaggio di Parametri nelle Funzioni

```
//esempio di funzioni con parametri passati con riferimenti  (by reference)

#include <iostream>
using namespace std;

int addition (int & x, int & y); // possibile anche int addition (int&, int&); Sto dicendo al
                                 compilatore che sto passando gli argomenti per riferimento
//& e' detto operatore di riferimento (reference operator )

int main () {
  int z, x=3, y=5;
  int & xx = x;//  xx e' un alias per x (sono nomi diversi della stessa cosa)
  int & yy = y;

  cout << " x = " << x << "  y = " << y << endl;
  cout << " xx = " << xx << "  yy = " << yy << endl;
  z= addition (x,y);  // il prototipo ha  avvertito il compilatore che i parametri sono
                      //passati  per riferimento
  cout << " x = " << x << ", y = " << y << " and  the sum is " << z << endl;
}
```

Passaggio di Parametri nelle Funzioni

```
int addition (int& a, int& b)
{
  int r;
  r = a + b;
                    // voglio vedere cosa succede se scambio in questo scope  a con b
  int temp =a;
  a = b;            // sto scambiando gli alias delle variabili x=3 e y=5 della funzione main
  b = temp;      // e quindi sto scambiando proprio i valori delle due variabili
  return (r);
}
```

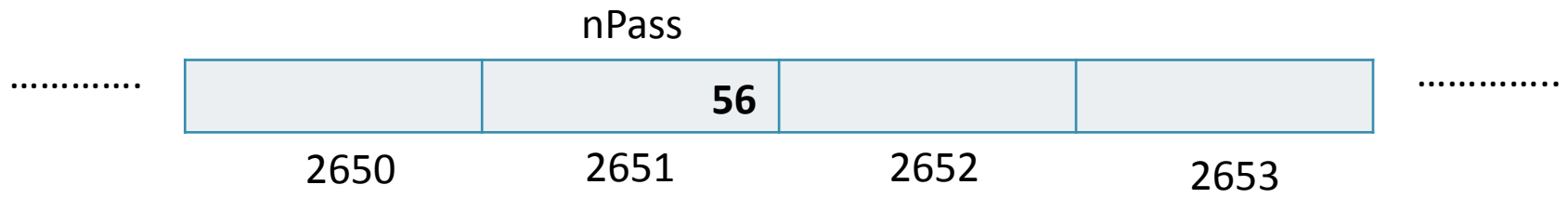
==========

```
g++ -o FunzioneByReference FunzioneByReference.cc

./FunzioneByReference
 x = 3  y = 5
 xx = 3  yy = 5
 x = 5, y = 3 and  the sum is 8
```

Indirizzi

- La memoria del computer e' pensabile come una successione di celle che sono numerate in modo progressivo. La minima dimensione della cella e' un byte



nPass=56;

- Alla variabile nPass e' stato assegnato il valore 56 e questa variabile ha come indirizzo di memoria 2651 (l'indirizzo non puo' essere assegnato da una istruzione. Esso viene predeterminato dal programma).

- Operatore unario di indirizzo (reference operator) & : restituisce la locazione di memoria dell'operando

- Indi = & nPass; (la variabile Indi ha come valore 2651 che e' l'indirizzo nella memoria di nPass

Puntatori

- Una variabile di tipo puntatore contiene l'indirizzo di un'altra variabile (detta variabile puntata). Ad ogni tipo di variabile corrisponde un particolare tipo di puntatore!

```
double *  p;  // * mi dice che p e' una variabile di tipo puntatore (ad un double)
double * p1, * p2;
```

- L'operatore di dichiarazione * va ripetuto in quanto sintatticamente e' da considerare un prefisso dell'identificatore e non un suffisso del tipo.

- Operatore di dereferenziazione * : applicato ad un puntatore restituisce il valore della variabile puntata. Considerando l'esempio precedente, si ha che *indi vale 56

- Gli operatori di riferimento & e di dereferenziazione * hanno effetto opposto: nPass == * &nPass

N.B. : l'operatorio unario di dichiarazione * NON va confuso con l'operatore di dereferenziazione per il quale si usa lo stesso simbolo *

Puntatori

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


int main() {
  int firstvalue, secondvalue;
  int * mypointer;

  mypointer = &firstvalue;  // inizializzo mypointer con l'indirizzo di memoria di firstvalue
  *mypointer = 10;  // assegno 10 al valore puntato da mypointer (anche firstvalue  e' 10)
  mypointer= &secondvalue; //assegno a mypointer l'indirizzo di memoria di secondvalue
  *mypointer = 20;
  cout<< "firstvalue  =  "<< firstvalue  <<",  secondvalue  = "<<secondvalue<<endl;
  cout << "mypointer = " << mypointer << ",  &( *mypointer ) = " << &( *mypointer ) <<endl;
  return 0;
}
```

```
=====================================================
firstvalue = 10, secondvalue = 20
mypointer = 0xbffff6c4,  &( *mypointer ) = 0xbffff6c4
```

Passaggio di Parametri nelle Funzioni

```
//esempio di funzioni con parametri passati con puntatori  (by pointer)

#include <iostream>
using namespace std;

int addition (int * px, int * py); // possibile anche int addition (int*, int*); Sto dicendo al
                                   //compilatore che sto passando gli argomenti usando
puntatori


int main () {
  int z, x=3, y=5;
  int * xpointer, *ypointer;
  xpointer = &x;  //&  =  operatore di riferimento; sto inizializzando i puntatori
  ypointer = &y;

  cout << " x = " << x << "  y = " << y << endl;
  z= addition (xpointer, ypointer);  // il prototipo ha  avvertito il compilatore che i
                                     //parametri sono  passati  con puntatore
  cout << " x = " << x << ", y = " << y << " and  the sum is " << z << endl;
}
```

Passaggio di Parametri nelle Funzioni

```
int addition (int* a, int* b)
{
  int r;
  r = *a + *b;  // * operatore di dereferenziazione (da non confondere con la dichiarazione
                //di puntatore)
                // Voglio vedere ora cosa succede se scambio in questo scope  a con b
  int temp = *a;
  *a =  *b;           // sto scambiando i  valori puntati a  x=3 e y=5 della funzione main
  *b = temp;      // cioe' sto scambiando proprio i valori delle due variabili
  return (r);
}
```

==========

```
g++ -o FunzioneByPointer FunzioneByPointer.cc

./FunzioneByPointer
 x = 3  y = 5
 x = 5, y = 3  and  the sum is  8
```

Overload del Nome di una Funzione

- Funzioni che hanno diversa segnatura (cioe' diversi tipi di parametri e/o diverso numero di parametri) possono avere lo stesso nome nello stesso scope.

- Il compilatore a seconda della segnatura sa decidere qual e' la funzione giusta da utilizzare

- Se la segnatura della chiamata della funzione non coincide con nessuna delle funzioni dichiarate, allora il compilatore prova a promuovere il tipo dei parametri seguendo le regole della conversione automatica dei tipi; per esempio promuove un float a double oppure un char in int.

- Se anche cosi il compilatore non trova il match , allora esce con un errore scrivendo che la chiamata della funzione e' ambigua

Overload di Funzione

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

void Display (int x);
void Display (double x);
void Display (string  x);
void Display (int x, int y);

int main(){
  int a = 3, b=5;
  double c= 13.5;
  float d=12.2;
  string  mystring = "ciao";

  Display (a);
  Display (a,b);
  Display (c);
  Display (d);
  Display (mystring);
//  Display (mystring, mystring); //scommentare questa da un errore in compilazione
```

Overload di Funzione

```
 return 0;
}

void Display (int x) { cout << " The int is " <<  x << endl;}

void Display (double x) {cout << " The double is " <<  x << endl;}

void Display (string  x) { cout << " The string  is " <<  x  << endl;}

void Display (int x, int y) {
cout << " The first int is " <<  x << “; the second int is " << y << endl;
}
=========================================
g++ -o OverloadingNome OverloadingNome.cc
./OverloadingNome

 The int is 3
 The first int is 3; the second int is 5
 The double is 13.5
 The double is 12.2
 The string  is ciao
```

Namespace

Questa parola chiave (keyword) puo' essere usata per definire una scope. Questo e' particolarmente utile con programmi nei quali e' possibile che ci siano funzioni, oggetti o classi che hanno lo stesso nome.

Esempio di namespace:

```
namespace MyNameSpace {
int i ;
// dichiarazioni varie
}
```

Namespace definisce un suo scope. Quindi per accedere ad oggetti definiti all'interno di un namespace bisogna usare l'operatore di risoluzione della scope ::

Esempio se voglio usare i definita in MyNameSpace uso la forma:

```
MyNameSpace::i = 10;
```

Se i membri di un namespace sono usati spesso nel programma allora e' piu' comodo usare la direttiva using:

```
using namespace NomeDelNamespace;        oppure
using NomeDelNameSpace:: membro;
```

Namespace

```
include <iostream>
using namespace std;
//raggruppa tutte le quantita' (classi, funzioni, ecc ) della libreria standard di
// C++ sotto il nome std. Ci serve perche' usiamo cin, cout, setprecision.

int var=20;  //var globale

namespace myNamespace {
  int var = 5;
    int  square (int y){
    int l= y*y*y;  // qui sbaglio e calcolo il cubo di y !!!
      return l;
   }
}
int square ( int ); // prototipo della funzione square

int main() {
  cout << " var variabile globale  = " << var << endl;
  cout << "var globale =" << var << ", quadrato di var (globale) = " << square(var)<< endl;
  cout << " var appartenente al myNamespace  = " << myNamespace::var << endl;
```

Namespace

```
cout << "var globale =" << var << ", quadrato (funzione definita in  myNamespace) di
var (globale) = " << myNamespace::square(var)<< endl;
cout << "var globale =" << var << ", quadrato di var (definita in myNamespace) = ”
 << square(myNamespace::var)<< endl;

using myNamespace::var ;
cout << "var di myNamespace =" << var << ", quadrato di var (myNamespace) = " <<
square (var) << endl;
return 0;
}
int  square (int y)
{    return y*y;}
```

==========

```
g++ -o namespace namespace.cc
./namespace
var variabile globale  = 20
var globale =20, quadrato di var (globale) = 400
 var appartenente al myNamespace  = 5
var globale =20, quadrato (funzione definita in  myNamespace) di var (globale) = 8000
var globale =20, quadrato di var (definita in myNamespace) = 25
var di myNamespace =5, quadrato di var (myNamespace) = 25
```