Lezione 2

Richiami su : Puntatori, Array e Strutture

http://www.mi.infn.it/~palombo/didattica/Lab-TNDS/CorsoLab/LezioniFrontali/Intro2.pdf

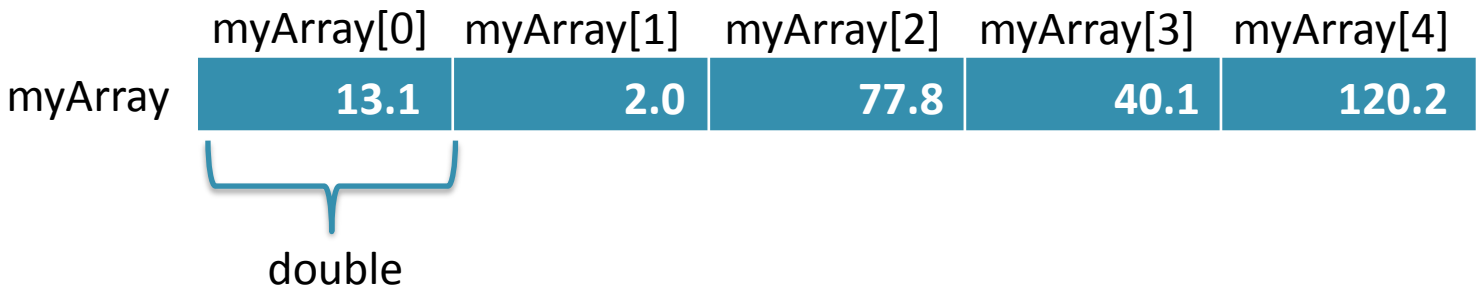
Fernando Palombo

Array

- Un insieme di n variabili dello stesso tipo poste in posizioni continue in memoria si dice array. Il tipo dell'array è quello delle sue variabili.

- Dichiarazione e inizializzazione di un array di double di 5 elementi:

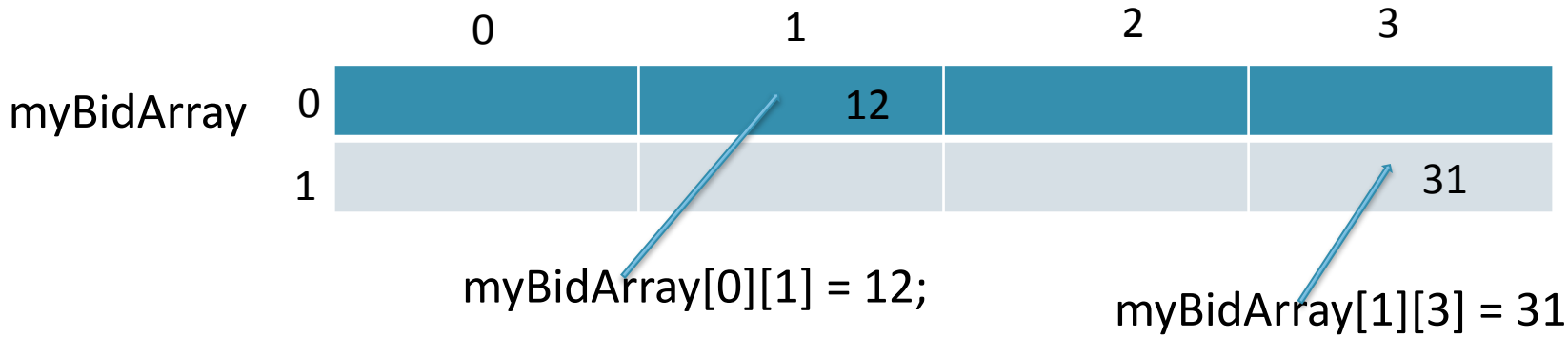
```
double  myArray [5] = {13.1, 2.0, 77.8, 40.1, 120.2};
```



Array

- double myArray [] = {13.1, 2.0, 77.8, 40.1, 120.2}; il compilatore deduce le dimensioni dell'array dal numero di valori inclusi nelle parentesi graffe.

- Array Multidimensionali : int myBidArray[2][4];



- Notare che : int myBidArray[2][4]; è equivalente a int myBidArray[8];

- In C++ è sintatticamente corretto uscire fuori dal range degli indici degli array. Nell'array myBidArray potremmo porre myBidArray[0][5] =21; senza avere errore di compilazione.

- Ovviamente sporcando zone di memoria out-of-range si possono avere runtime error. È un tipo di errore molto comune!!

Array come Parametri

```
//array as parameter in a function
#include <iostream>
#include <iomanip>
using namespace std;

void Display (int arg[], int length); //prototipo

int main(){
  int myArray[] = {13 , 21, 56, 88, 33};
  Display (myArray, 5);
  return 0;
}

void Display (int arg[], int length) {
  for (int n=0; n<length; n++) { cout << setw(5) << arg[n]; }
  cout << endl;
}
```

==========

```
g++ -o ArrayParameter ArrayParameter.cc
./ArrayParameter
 13   21   56   88   33
```

Puntatori e Array

- L'identificatore di un array è equivalente all'indirizzo del primo elemento dell' array. L' array quindi è un puntatore.

- L'array è però un puntatore speciale con qualche caratteristica in più :
  La dichiarazione di un puntatore comporta allocazione di memoria solo per la variabile puntatore ma non per la variabile puntata dal puntatore

- La dichiarazione di un array comporta invece l'allocazione di memoria sia per la variabile puntatore che per la variabile puntata:
  int * myArray[10]; : alloca memoria per il puntatore (COSTANTE) myArray
  alloca memoria per 10 valori di tipo int
  inizializza myArray con &myArray[0]
- Il puntatore è preso costante perché non si può cambiare l'indirizzo dell'array (mentre si può cambiare l'indirizzo di un normale puntatore)

```
int numbers[10];
int *p;
p= numbers;   // assegnazione valida
numbers = p; // errore  numbers  è un puntatore costante e non possiamo assegnargli
 un valore
```

Operatore di offset

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

int main(){
int numbers[5];
int * p;
p  = numbers;  *p = 10;
p++; *p = 20;
p = &numbers[2];   *p = 30;
p= numbers +3;    *p = 40;
p = numbers;    *(p+4) = 50;   //offset  di 4 ; assegno 50 alla variabile puntata da (p+4)

for ( int n=0; n<5; n++)  cout << numbers[n] << “,  “;
cout <<endl;
 return 0;
}
```

==========

```
g++ -o offset offset.cc
./offset
10,  20,  30,  40,  50,
```

Aritmetica con i Puntatori

Addizione e sottrazione sono le sole operazioni aritmetiche che hanno senso con i puntatori.

Queste operazioni hanno caratteristiche diverse a seconda del tipo di puntatore.

Supponiamo che un puntatore di tipo intero e che un intero ha dimensione di 4 byte : int * myint ; myint = 100

myint+n = myint + n*4 Quindi ad esempio myint contiene il valore 100 e myint++ contiene il valore 104

Se la nostra variabile avesse dimensione di 8 byte avremmo : myint = myint + n*8

Operatori di incremento (++) e di decremento (--) con puntatori

Questi operatori possono essere preposti o posposti alla variabile. Se sono preposti (ad esempio ++a) l'incremento è fatto prima che venga valutata l'intera espressione:

```
int a, b=2;
a= ++b -1;     // si ha che b=3 e a = 2
```

Se sono posposti prima si calcola l'intera espressione e dopo viene effettuato l'incremento:

```
int a,b=2;
a = (b++) -1 // si ha che a =1 , b=3
```

Si noti che questi operatori ++ e -- hanno priorità superiore a quella dell'operatore (*) di dereferenziazione . Quindi ad esempio *p++ è equivalente *(p++)

Essendo posposto l'operatore ++, allora viene incrementato il valore di p ma l'espressione è calcolata col valore puntato da p prima che venisse incrementato!

(*p)++ Qui invece si ha il valore puntato da p aumentato di 1. Il valore del puntatore non viene alterato.

Operatori (++) e (--) con puntatori

```
include <iostream>
using namespace std;

int main(){
  int numbers[]= {10,20,30,40,50};
  int *p = numbers;
  cout <<“Valore del puntatore = " << p <<endl;
  int z = *p++;
  cout << "*p++ = " << z << ”. Valore del puntatore =  " << p <<  endl;
  cout << "(*p)++ = " << (*p)++  << ”. Valore del puntatore =  " << p <<  endl;
}

=====================================
valore del puntatore = 0xbffff630
*p++ = 10.  Valore del puntatore =  0xbffff634
(*p)++ = 20. Valore del puntatore =  0xbffff634
```

Puntatore a Funzione

- In C++ sono usati anche puntatori a funzione. Si usano in genere per passare una funzione come argomento ad un'altra funzione.

int function (int , int); prototipo della funzione
int (*function)(int , int); dichiarazione di puntatore a function . Esempio:

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

int addition (int a, int b);
int subtraction (int a, int b);
int operation (int x, int y, int (*functocall) (int, int));

int main() {
int m,n;
int (*minus)(int,int) = subtraction;// minus puntatore alla funzione subtraction
m = operation (7, 5, addition);
cout<<"Da addition: m = 7 + 5 = " << m <<endl;
n = operation (m, 2, subtraction);
cout<<"Da subtraction:  m - 2 = " << n <<endl;
n = operation (20, m, minus);
cout<<"Da minus: 20 - m  = " << n <<endl;
```

Puntatore a Funzione

```
 return 0;
}

int addition (int a, int b){  return (a+b);}
int subtraction (int a, int b){  return (a-b);}
int operation (int x, int y, int (*functocall) (int, int))
 { int g;
   g = (*functocall)(x,y); //puntatore alla funzione functocall
   return (g);
 }
```

==========

```
g++ -o PuntatoreFunzione2 PuntatoreFunzione2.cc
./PuntatoreFunzione2

Da addition: m = 7 + 5 = 12
Da subtraction:  m - 2 = 10
Da minus: 20 - m  = 8
```

Attenzione ai Puntatori

- I puntatori sono utili in molte applicazioni ma se accidentalmente un puntatore ha un valore sbagliato , può essere molto difficile scoprire dove si trova l'errore.

- Se per esempio il puntatore punta in una zona sbagliata cambiandone il valore è molto probabile che l'alterazione si manifesti parecchio dopo durante l'esecuzione e si cerca l'errore in una parte sbagliata del programma.

Puntatori non inizializzati:

```
// Questo programma è sbagliato!
int main(){
int x, *p;
x=10;
*p = x;   dove punta p?????
return 0;
}
```

Attenzione ai Puntatori

Confronto non valido tra puntatori

Confronto tra puntatori che non accedono allo stesso array, non è generalmente valido e causa spesso errori:

```
char s[80];
char y[80];
char *p1, *p2;
p1=s;
p2=y;
If (p1 < p2)  ….
```

Sbagliato: C++ non garantisce in quale parte della memoria vengono messe le variabili! Il codice deve funzionare indipendentemente dal punto della memoria dove vengono memorizzati i dati

Memoria Dinamica:operatore new

- In fase di scrittura del programma la memoria disponibile è quella dichiarata per le variabili usate nel codice sorgente.

- Spesso però solo eseguendo il programma (cioè durante il runtime) si scopre di aver bisogno di un altro oggetto (altra variabile o array di dimensioni maggiori di quelle dichiarate). L'allocazione (dinamica) di memoria durante il runtime in C++ è fatta tramite gli operatori new e delete:

```
int * ptr;
ptr = new int [5];
```

- Qui ho assegnato dinamicamente memoria per 5 elementi di tipo int. ptr punta al primo di questi 5 elementi. Quindi si accede al primo elemento con ptr [0] (oppure *ptr); si accede al secondo elemento con ptr [1] (oppure *(ptr +1)) , ecc

- Quindi si accede a questa memoria dinamica per dereferenziazione del puntatore (che contiene il suo indirizzo)

Memoria heap

L'allocazione della memoria dinamica è fatta in un'area di memoria detta heap

L'area allocata non è identificata da un nome ed è accessibile tramite dereferenziazione di un puntatore

Il suo scope coincide con quello del puntatore che contiene il suo indirizzo.

La lifetime di questa area allocata coincide con quella dell'intero programma.

Se il puntatore esce dallo scope, l'area allocata non è più accessibile e quindi resta occupata sino alla fine del programma :

Errore di memory leak (molto , molto comune!!!)

Per evitare errori di memory leak bisogna deallocare l'area allocata (e prima che si esca fuori dallo scope dove c'è il puntatore che l'ha allocata!)

Poiché la memoria heap è limitata, questa può essere esaurita e quindi è bene verificare che l'assegnazione dinamica di memoria e' stata fatta con successo.

Memoria Dinamica: operatore delete

```
#include <iostream>
using namespace std;
int main(){
int *p;
 p = new int (100); // alloca un intero e lo inizializza a 100
 if(!p)  {//new ritorna un valore 0 se fallisce ad allocare la memoria
  cout<< "allocation error" << endl;
  exit(1);
 }
 cout << "At "<<p <<  "";
 cout << " there is the value " << *p << endl;
 delete p;
 return 0;
}
```

Memoria Dinamica: operatore delete

```
#include <iostream>
using namespace std;
int main(){
  int *p, i;
 p = new int [20]; // alloca un array di 20 interi
 if(!p)  {
  cout<< "allocation error" << endl;
  exit(1);
 }

for(i=0; i<20; i++)
p[i]=i;

 for(i=0; i<20; i++) {cout << p[i] << "  ";}
 cout <<endl;
 delete [] p; // le parentesi quadre servono per liberare la memoria
                  // occupata da un array
 return 0;
}
```

Strutture

La struttura permette di raggruppare sotto un singolo nome un insieme di variabili. Viene dichiarata usando la parola chiave struct. Ogni variabile nella struttura è detta membro della struttura (o campo o elemento)

Esempio di dichiarazione di struttura:

```
struct   Frutta {
  int     calorie;
  double  costo;
  double peso;
  ……;
} pera, anguria;   //facoltativo;  indica oggetti di tipo struct . Notare il ; finale
```

Dichiarazione di oggetto di tipo Frutta:

```
Frutta banana;
```

Si accede ai membri della struttura tramite l'operatore .

```
banana.costo =  5;  // assegniamo 5 al costo dell'oggetto banana
 cout << banana.cost // stampiamo sullo schermo il costo della banana
```

Puntatori a Strutture

Come ogni altro tipo, anche una struttura ha un puntatore (tipo struttura) che la punta.

Frutta banana; // oggetto di tipo Frutta
Frutta * p_banana;
p_banana = &banana; //puntatore alla struttura banana

Usando i puntatori , si accede ai membri di una struttura con l'operatore ->

p_banana->peso = 3;

Questo è equivalente a: (*p_banana).peso = 3;

Array di strutture:
Frutta canestro[20]; // array di 20 oggetti (strutture) di tipo Frutta

cout< canestro[1].costo; // scrivo il membro costo della seconda struttura (l'array inizia con indice zero!)

Esempio di Struttura

```
// Crea e usa la struttura  Time
#include <iostream>
using namespace std;
#include <iomanip>

// structure definition
 struct Time {
   int hour; // 0-23 (24-hour clock format)
   int minute; // 0-59
   int second; // 0-59
 };

 void printUniversal( const Time & ); // prototype
 void printStandard( const Time & ); // prototype

int main() {
  Time dinnerTime;
  dinnerTime.hour = 18; // set hour member of dinnerTime
  dinnerTime.minute = 30; // set minute member of dinnerTime
```

Esempio di Struttura

```
dinnerTime.second = 0; // set second member of dinnerTime

cout << "Dinner will be held at ";
 printUniversal( dinnerTime );//struttura passata per riferimento ad una funzione

 cout << " universal time,\nwhich is ”;
 printStandard( dinnerTime );
 cout << " standard time.\n";
 cout << endl;
 return 0;
}

// print time in universal-time format
 void printUniversal( const Time &t )
 {
 cout << setfill( '0' ) << setw( 2 ) << t.hour << ":"
  << setw( 2 ) << t.minute << ":"
   << setw( 2 ) << t.second;
 }
```

Esempio di Struttura

```
// print time in standard-time format
 void printStandard( const Time &t ) {
  cout << ( ( t.hour == 0 || t.hour == 12 ) ?
            12 : t.hour % 12 ) << ":" << setfill( '0' )
            << setw( 2 ) << t.minute << ":"
            << setw( 2 ) << t.second
            << ( t.hour < 12 ? " AM" : " PM" );
 }
```

==========

```
g++ -o Time Time.cc
./time
Dinner will be held at 18:30:00 universal time,
which is 6:30:00 PM standard time.
```