Programmazione 2
e Laboratorio di Programmazione

Corso di Laurea in

Informatica

Università degli Studi di Napoli "Parthenope"

Anno Accademico 2023-2024

Prof. Luigi Catuogno

1

Informazioni sul corso

Docente	Luigi Catuogno **luigi.catuogno@uniparthenope.it**
Orario	Lun: 9:00-11:00 Mer: 11:00-13:00
Sede	Centro Direzionale Napoli **Aula Magna**
Ricevimento	Mer: 14:00-16:00 (previo appuntamento) Ufficio docente oppure Team: **cxxa3bo**

2

Libri di testo

Introduzione al linguaggio – costrutti e tecniche di base

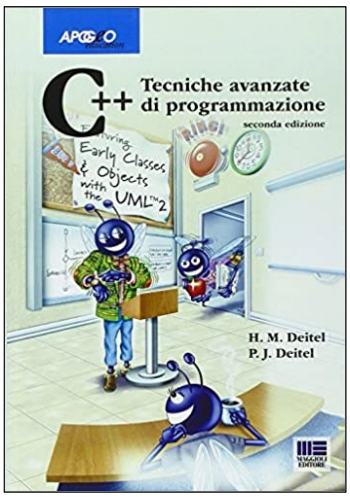
[FdP] H. M. Deitel, P. J. Deitel
C++ Fondamenti di programmazione
II ed. (2014) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8571-9

3

Libri di testo

Tecniche avanzate e strutture dati elementari

[TAP] H. M. Deitel, P. J. Deitel
C++ Tecniche avanzate di programmazione
II ed. (2011) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8572-6



4

Risorse on-line

Team del corso

Programmazione 2 AA 2023-24 - Prof. Catuogno
Comunicazioni, incontri e avvisi per il corso
Codice: **ftomzjx**

Piattaforma e-learning

Programmazione II e Laboratorio di Programmazione II - A.A. 2023-24
Materiale didattico, manualistica, esercitazioni.
URL: https://elearning.uniparthenope.it/course/view.php?id=2386

5

Dal C al C++

6

Riepilogo su array e puntatori

7

Ripasso su Array e Matrici

Struttura dati composta da un insieme di elementi dello stesso tipo

Ciascun elemento è identificato univocamente mediante un indice (un numero intero che ne definisce la posizione nell'array)

La lunghezza di un array è stabilita al **momento della sua dichiarazione** (e non cambia)

In un array lungo N, troveranno posto N elementi del tipo base che saranno identificati con gli indici da 0 a n-1. Indicare un indice al di fuori di questo intervallo è un errore *(non sempre segnalato dal compilatore…)*

8

Ripasso su Array e Matrici

Dichiarazione di un array:

tipo_base ***identificatore_array*** **[***dimensione(int)***]**;

```
int serie[10];
float v1[4],v2[4];
bool x[2],z;
char c[12];
```

9

Ripasso su Array e Matrici

Espressioni con gli elementi di un array:

```
x[0]=true;
x[1]=false;
z=x[0] && x[1] && x[2];
```

```
for(int i=1;i<=10;i++)
      cin >> serie[i-1];
```

L'indice può anche essere il risultato di una espressione *a valori interi* purché sia garantito che il risultato indichi una posizione compresa tra 0 e N-1 (dove N è la lunghezza dell'array)

10

Ripasso su Array e Matrici

Si possono avere array *multidimensionali:*

tipo identificatore **[***dim*$_1$**][***dim*$_2$**] … [***dim*$_n$**];**

```
int tabella[12][2];
float m1[4][4],m2[4][5][6];
```

In un array a N dimensioni, ogni elemento è identificato da una N-pla di indici che ne rappresentano le «coordinate» all'interno dell'array.

11

Ripasso su Array e Matrici

Dichiarazione con inizializzazione

```
int vect[3]={0,1,0};
float m1[2][3]= {{1,3,4},{4,5,6}};
float m2[2][3]= {1,2,3,4,5,6};
int tabella[12][2]={0};
```

Gli array sono memorizzati una riga alla volta, pertanto m1 e m2 sono inizializzate con i medesimi valori

Se nell'inizializzazione si specifica il valore di un numero di elementi minore delle posizioni dichiarate, tutti gli elementi rimanenti saranno inizializzati a zero.

12

Esempio: operazioni con i vettori

Prodotto *scalare* di due vettori di 3 elementi:

$$s = v \cdot w = \sum_{i=0}^{2} v_i w_i$$

13

Esempio: operazioni con i vettori

Prodotto *scalare* di due vettori di 3 elementi:

$$s = v \cdot w = \sum_{i=0}^{2} v_i w_i$$

```
float s=0, v[3], w[3];
…
s=(v[0]*w[0] +  v[1]*w[1]+ v[2]*w[2]);
```

14

Esempio: operazioni con i vettori

```
10  int main()
11  {
12          float v[3]={1,1,6}, w[3]={2,1,-1}, s=0;
13          cout << "Prodotto scalare:"<<endl;
14          cout << "( ";
15          for (int i=0;i<3;i++) {
16                  cout << v[i] <<" ";
17                  s=s+v[i]*w[i];
18          }
19          cout << ") . ( ";
20          for (int i=0;i<3;i++)
21                  cout << w[i] <<" ";
22          cout << ") = "<< s << endl;
23  }
```

15

Esercizio: la morra cinese

La Morra Cinese è un gioco antichissimo già attestato in Cina ai tempi della Dinastia Han (206 AC – 220 DC). Se ne contano innumerevoli varianti ed è stato oggetto di centinaia di studi scientifici (delle più varie discipline).

Scriviamo un programma C++ che implementa una versione in cui un giocatore umano affronta il calcolatore per un numero di round fissato all'inizio. Al giocatore che prevale in ciascun round, viene assegnato un punto. Nessun punteggio viene attribuito in caso di parità. Completati tutti i round, il programma visualizza i punteggi e termina.

16

Esercizio: la morra cinese

Suggerimenti: Al di là della gestione dell'interazione con l'utente, il programma deve essenzialmente codificare la relazione *batte* tra le diverse combinazioni di mosse che possono verificarsi, secondo la seguente tabella:

batte	*sasso*	*carta*	*forbici*
sasso	*Falso*	*Falso*	*Vero*
Carta	*Vero*	*Falso*	*Falso*
forbici	*Falso*	*Vero*	*Falso*

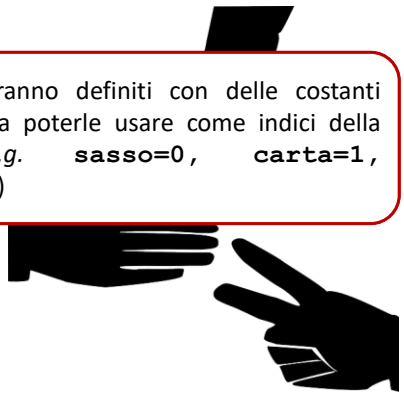
17

Esercizio: la morra cinese

Suggerimenti: Al di là della gestione dell'interazione con l'utente, il programma deve essenzialmente codificare la relazione *batte* tra le diverse combinazioni di mosse che possono verificarsi, secondo la seguente tabella:

batte	*sasso*	*carta*	*forbici*
sasso	*Falso*	*Falso*	*Vero*
Carta	*Vero*	*Falso*	*Falso*
forbici	*Falso*	*Vero*	*Falso*

I «gesti» saranno definiti con delle costanti intere così da poterle usare come indici della tabella (*e.g.* `sasso=0`, `carta=1`, `forbici=2`)



18

Esercizio: la morra cinese

Suggerimenti: Al di là della gestione dell'interazione con l'utente, il programma deve essenzialmente codificare la relazione *batte* tra le diverse combinazioni di mosse che possono verificarsi, secondo la seguente tabella:

batte	*sasso*	*carta*	*forbici*
sasso	*Falso*	*Falso*	*Vero*
Carta	*Vero*	*Falso*	*Falso*
forbici	*Falso*	*Vero*	*Falso*

I «gesti» saranno definiti con delle costanti intere così da poterle usare come indici della tabella (*e.g.* **`sasso=0`**, **`carta=1`**, **`forbici=2`**)

Il giocatore n.1 (il computer) produce casualmente la sua mossa *m1*, il giocatore n.2 (l'utente) inserirà la sua mossa *m2* da tastiera.

19

Esercizio: la morra cinese

Suggerimenti: Al di là della gestione dell'interazione con l'utente, il programma deve essenzialmente codificare la relazione *batte* tra le diverse combinazioni di mosse che possono verificarsi, secondo la seguente tabella:

batte	*sasso*	*carta*	*forbici*
sasso	*Falso*	*Falso*	*Vero*
Carta	*Vero*	*Falso*	*Falso*
forbici	*Vero*	*Falso*	*Falso*

I «gesti» saranno definiti con delle costanti intere così da poterle usare come indici della tabella (*e.g.* **`sasso=0`**, **`carta=1`**, **`forbici=2`**)

Il giocatore n.1 (il computer) produce casualmente la sua mossa *m1*, il giocatore n.2 (l'utente) inserirà la sua mossa *m2* da tastiera.

La mossa *m1* prevale sulla mossa *m2* se `batte[m1][m2]` è *true*. Vice versa, *m2* prevale su *m1* se `batte[m2][m1]` è *true*. La parità si verifica quando entrambe le celle della tabella contengono *false*.

20

Esercizio: la morra cinese

```
 1  #include<iostream>
 2  #include<cstdlib>
 3  #define NUMGESTI 3
 4  using namespace std;
 5
 6  const int SASSO=0,CARTA=1,FORBICI=2;
 7
 8  bool batte[3][3]{{false,false,true},
 9                   {true,false,false},
10                   {false,true,false}};
11
12  char nomeG[3][8]={"sasso","carta","forbici"};
```

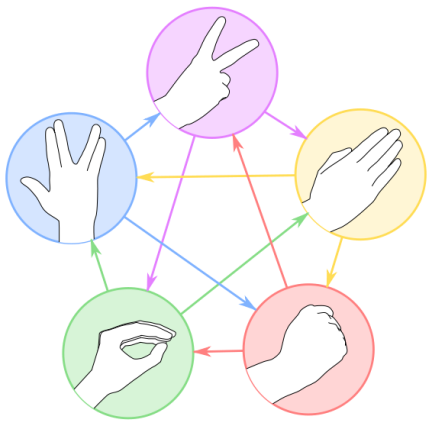
21

Esercizio: la morra cinese

- Una variante della morra cinese, è stata inventata da Karen Bryla e Sam Kass negli anni 2000.
  - Alla morra tradizionale sono stati aggiunti la lucertola e Spock. Le relazioni tra i simboli sono riassunte di seguito e nel grafico al lato

Forbici tagliano carta; carta avvolge sasso; sasso rompe forbici; forbici decapitano lucertola; lucertola mangia carta; carta smentisce Spock; Spock vaporizza sasso; sasso schiaccia lucertola; lucertola avvelena Spock; Spock rompe forbici.

http://www.samkass.com/theories/RPSSL.html



24

Ripasso su Array e Matrici

E' possibile passare un array come parametro a una funzione. Tuttavia, occorre ricordare che questi sono passati sempre *per indirizzo*. Inoltre, occorre sempre passare alla funzione anche la lunghezza dell'array.

```
bool is_sorted(int sequence[], int len) {…}
```

```
main()
{
      int numbers[5]={1,56,4,56,1};
      cout << is_sorted(numbers,5);
```

25

Esempio: *sequenza ordinata?*

Restituisce true se gli interi contenuti nell'array **sequence** sono ordinati in maniera crescente.

```
 1  #include<iostream>
 2  using namespace std;
 3
 4  bool is_sorted(int sequence[], int len)
 5  {
 6
 7
 8        for (int i=1;i<len;i++)
 9              if (sorted[i]<sorted[i-1])
10                    return false;
11        return true;
12  }
13
```

26

Esercizio: *sequenze palindrome?*

Si scriva un programma in C++ che chieda all'utente di inserire una sequenza di interi in un array di 8 elementi e dica se tale sequenza è *palindroma.*

27

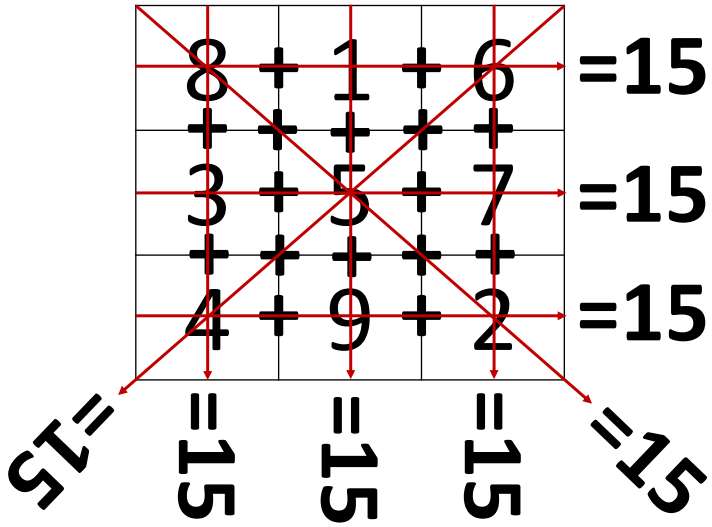
Esercizio: *Il quadrato magico…*

Un quadrato magico è una matrice di interi positivi $N \times N$ riempito con i numeri da 1 a N^2 disposti in modo tale che la somma dei numeri che occupano ciascuna riga, o colonna o diagonale, dia sempre lo stesso valore. Tale valore è costante e dipende dall'ordine del quadrato.

8	1	6
3	5	7
4	9	2

Quadrato magico di ordine 3.
Il numero magico è 15

33



34

Esercizio: *Il quadrato magico…*

- Si scriva il programma C++ che:
  - Chieda all'utente di riempire una matrice di dimensioni 3x3 con i numeri da 1 a 9 (scelti ciascuno una sola volta)
  - Dica se la matrice inserita è un quadrato magico;

Attenzione: Il programma deve essere realizzato in modo che, modificando al più due variabili (quella che indica l'ordine e quella che indica il numero magico), possa essere utilizzato per quadrati di ordine arbitrario.

35

Esercizio: *Il quadrato magico…*

4	9	2
8	1	6
3	5	7

NO

2	9	4
7	5	3
6	1	8

SI

36

Esercizio: *Il quadrato magico…*

Modificando opportunamente i parametri **`ordine`** e **`numero`**, è possibile utilizzare il programma per verificare qualsiasi quadrato. Il numero magico funzione dall'ordine.

Si modifichi il programma per eseguire la verifica di un quadrato di ordine 5. In questo caso, il numero magico è 65.

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

46

Array e puntatori

47

Array e puntatori

Array e puntatori sono strettamente correlati in C++ e possono essere utilizzati in maniera *quasi* equivalente;

- Il nome di un array è in realtà un puntatore *costante*
- I puntatori possono essere utilizzati in tutte le operazioni che coinvolgano gli indici di un array *(anche con la stessa sintassi)*

48

Array e puntatori

Il nome di un array è in realtà un puntatore *costante* che punta al suo primo elemento

```
float farr[100]={0.1} // il resto tutto a 0
float *fPtr;

fPtr=farr;
cout << *fPtr << endl;
```

```
0.1
```

49

Esempio: *array e puntatori #1*

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6          const int MAX=3;
7          int arr[MAX]={1,2,3}, *arrPtr;
8
9          arrPtr = arr;
10
11         cout << "arr[0]=" << arr[0]<<endl;
12         cout << "*arrPtr=" << *arrPtr << endl;
13         cout << "*arr=" << *arr <<endl;
14
15 }
```

```
arr[0]=1
*arrPtr=1
*arr=1
```

50

Array e puntatori

Mediante l'*aritmetica dei puntatori*, è possibile ispezionare una struttra dati (*e.g.* un array)

```
float farr[100]={0.1,4,1.6} //il resto
float *fPtr;

fPtr=farr;
cout << "farr[1]=" <<  *(fPtr+1) << endl;
cout << "farr[2]=" <<  *(fPtr+2) << endl;
```

Per raggiungere il secondo elemento (**farr[i]**) sommiamo **i** a **fPtr**. La quantità **i** è detta *offset*

```
4
1.6
```

51

Esempio: *array e puntatori #2*

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6         const int MAX=3;
7         int main_array[MAX]={1,2,3}, *arrayPtr;
8         arrayPtr=main_array;
9
10        for (int i=0;i<MAX;i++) {
11              cout << "arrayPtr+" << i << "=" << *arrayPtr << endl;
12              arrayPtr++;
13        }
14        cout << endl;
15 }
```

Qui, ogni volta, **arrayPtr** è incrementato di un offset pari a 1

52

Esempio: *array e puntatori #3*

```
1  #include<iostream>
2  using namespace std;
3
4  void somma10(int *fun_array, int size){
5          for(int i=0;i<size;i++){
6                  *fun_array+=10;
7                  fun_array++;
8          }
9  }
10 int main()
11 {
12         const int MAX=3;
13         int main_array[MAX]={1,2,3};
14         somma10(main_array,MAX)
15         for (int i=0;i<MAX;i++)
16                 cout << "main_array["<<i<<"]="<<main_array[i]<<endl;
17 }
```

Chiamo la funzione **somma10**, passandole l'indirizzo base dell'array e la sua lunghezza.

53

Esempio: *array e puntatori #3*

Inizialmente **fun_array**, punta alla *base* dell'array (al primo elemento);

```
1  #include<iostream>
2  using namespace std;
3
4  void somma10(int *fun_array, int size){
5          for(int i=0;i<size;i++){
6                  *fun_array+=10;
7                  fun_array++;
8          }
9  }
10 int main()
11 {
12         const int MAX=3;
13         int main_array[MAX]={1,2,3};
14         somma10(main_array,MAX)
15         for (int i=0;i<MAX;i++)
16                 cout << "main_array["<<i<<"]="<<main_array[i]<<endl;
17 }
```

Ad ogni iterazione sommiamo 10 al contenuto dell'elemento dell'array puntato da **fun_array;**

«spostiamo» **fun_array** sull'elemento successivo…

54

Aritmetica dei puntatori

Operazioni consentite con gli operatori

```
float *fPtr;
```

Incremento e decremento (*offset = 1*) prefissi o postfissi;

```
++fPtr; --fPtr;
fPtr++; fPtr--;
```

Occorre fare attenzione a che l'indirizzo puntato non oltrepassi limiti inferiore e superiore della struttura dati «spazzata» da fPtr; (*e.g.* per gli array, fPtr non deve essere minore dell'indirizzo base, né superiore all'indirizzo dell'ultimo elemento;

55

Aritmetica dei puntatori

Operazioni consentite con gli operatori

```
int i=1;
float *fPtr;
```

Somma di costanti o espressioni a valori interi;

```
fPtr=fPtr-1;
fPtr+=2*i;
```

Occorre fare attenzione a che l'indirizzo puntato non oltrepassi limiti inferiore e superiore della struttura dati «spazzata» da fPtr; (*e.g.* per gli array, fPtr non deve essere minore dell'indirizzo base, né superiore all'indirizzo dell'ultimo elemento;

56

Precedenza tra gli operatori

Priorità

maggiore

minore

Operatore	Descrizione
`(), [], .(punto), ->`	funzione, accesso array, struct, union e classi
`++, --, …`	unari incremento postfissi
`++, --, &, *, +, -, !`	unari incremento *(de)*referencing, prefissi, segno algebrico, not logico
`*, /, %`	binari moltiplicativi
`+, -`	binari additivi
`<<, >>`	inserzione/estrazione
`< <=, >, >=`	relazionali
`==, !=`	uguaglianza
…	…

57

Precedenza tra gli operatori

Priorità

maggiore

minore

Operatore	Descrizione
`&&`	and logico
`||`	or logico
`?:`	condizionale ternario
`=, +=, -=, …`	assegnamento e composti
`,`	virgola

58

Aritmetica dei puntatori

Per esempio nelle espressioni

```
x = *fPtr++;
```

Prima c'è l'incremento del puntatore e poi il dereferenziamento, **x** è una variabile di tipo **float**

```
*fPtr+=1; *fPtr=*f2Ptr;
```

Il dereferenziamento viene prima degli assegnamenti. Qui si incrementa di 1 il float «puntato», non il puntatore. L'assegnamento è tra i due float puntati.

59

Aritmetica dei puntatori

Per esempio nelle espressioni

```
(*fPtr)++;
```

Incrementa la variabile puntata da **fPtr;**

```
*fPtr+=1; *fPtr=*f2Ptr;
```

Il deferenziamento viene prima degli assegnamenti. Qui si incrementa di 1 il float «puntato», non il puntatore. L'assegnamento è tra i due float puntati.

60

Aritmetica dei puntatori

Supponiamo di avere un insieme di interi memorizzati in un'area contigua di memoria (*e.g.* un array), sommare (sottrarre) a un puntatore un certo offset *i*, significa: «far sì che esso punti un intero *i* posizioni più in avanti (indietro).

Tuttavia, questa operazione si traduce in un incremento (decremento) dell'indirizzo contenuto dal puntatore di *i volte* la taglia del tipo base, nel nostro caso **`int`**.

61

L'operatore **`sizeof()`**

L'operatore **`sizeof()`** restituisce il numero di byte necessari per memorizzare la variabile o il tipo indicati tra parentesi:

Tipo/variabile	Espressione	Ris.
`char`	`sizeof(char)`	1
`int`	`sizeof(int)`	4
`double`	`sizeof(double)`	8
`char x[8]`	`sizeof(char[8]) o sizeof(x)`	8
`Struct {double y;double z;} k;`	`sizeof(k)`	16
`int*`	`Sizeof(int*)`	8

62

Esempio: *aritmetica degli operatori*

```
 4  int main()
 5  {
 6          const int MAX=5;
 7          int inarr[MAX]={1,2,3,4,5},*inarrPtr=inarr;
 8          double dbarr[MAX]={0.1,0.2,0.3,0.4,0.5}, *dbarrPtr=dbarr;
 9
10          cout << "inarrPtr="<< inarrPtr << ", *inarrPTR="<<(*inarrPtr) << endl;
11          cout << "dbarrPtr="<< dbarrPtr <<          *dbarrPTR="<<(*dbarrPtr) << endl;
12
13          inarrPtr++;
14          dbarrPtr++;
15
16          cout << "inarrPtr="<< inarrPtr << ", *inarrPTR="<<(*inarrPtr) << endl;
17          cout << "dbarrPtr="<< dbarrPtr << ", *dbarrPTR="<<(*dbarrPtr) << endl;
18  }
```

Visualizzo l'indirizzo contenuto dal puntatore e il dato che vi è contenuto;

63

Esempio: *aritmetica degli operatori*

```
inarrPtr=0x6ffde0, *inarrPTR=1
dbarrPtr=0x6ffdb0, *inarrPTR=0.1
inarrPtr=0x6ffde4, *inarrPTR=2
dbarrPtr=0x6ffdb8, *dbarrPTR=0.2
```

```
 4  int main()
 5  {
 6          const int MAX=5;
 7          int inarr[MAX]=
 8          double dbarr[MAX
 9
10          cout << "inarrPtr
11          cout << "dbarrPtr
12
13          inarrPtr++;
14          dbarrPtr++;
15
16          cout << "inarrPtr="<< inarrPtr << ", *inarrPTR="<<(*inarrPtr) << endl;
17          cout << "dbarrPtr="<< dbarrPtr << ", *dbarrPTR="<<(*dbarrPtr) << endl;
18  }
```

Gli indirizzi dei puntatori a intero, aumentano di 4 byte per ogni unità di offset, poiché sizeof(int)=4;

64

Esempio: *aritmetica degli operatori*

```
inarrPtr=0x6ffde0, *inarrPTR=1
dbarrPtr=0x6ffdb0, *dbarrPTR=0.1
inarrPtr=0x6ffde4, *inarrPTR=2
dbarrPtr=0x6ffdb8, *dbarrPTR=0.2
```

```
 4  int main()
 5  {
 6          const int MAX=5
 7          int inarr[MAX]=
 8          double db
 9
10          cout << "inarrPt
11          cout << "dbarrP
12
13          inarrPtr++;
14          dbarrPtr++;
15
16          cout << "inarrPtr="<< inarrPtr << ", *inarrPTR="<<(*inarrPtr) << endl;
17          cout << "dbarrPtr="<< dbarrPtr << ", *dbarrPTR="<<(*dbarrPtr) << endl;
18  }
```

Gli indirizzi dei puntatori a double, aumentano di 8 byte per ogni unità di offset, poiché sizeof(double)=8;

65

Esercizio: *sequenze palindrome coi puntatori*

Scrivere un programma C++ che chieda all'utente di riempire un array di 5 interi e verifichi se la sequenza inserita è palindroma. Nello svolgimento:

La verifica deve essere effettuata da una funzione che prende il puntatore all'array e la sua lunghezza e restituisce un bool.

Nella funzione, lo scorrimento dell'array deve essere effettuato con dei puntatori.

66

Esercizio: *sequenze palindrome coi puntatori*

```
23  int main()
24  {
25          const int MAX=3;
26          int main_array[MAX]={1,2,1};
27
28          cout << "isPalindrome=" << isPalindrome(main_array,MAX) << endl;
29  }
```

67

Esercizio: *numeri uguali in posti uguali*

Scrivere un programma C++ che chieda all'utente di riempire due array di 5 interi calcoli quante volte, nei due array compare lo stesso numero nella stessa posizione.

Per esempio se **`arr1={1,12,54,79,39}`** e **`arr2={1,79,47,81,39}`** il risultato è 2 (1 e39)

Il conteggio deve essere effettuata da una funzione che prende il puntatore ai due array e la loro lunghezza e restituisca un intero.

Nella funzione, lo scorrimento dell'array deve essere effettuato utilizzando l'aritmetica dei puntatori

69

Esercizio: *al contrario*

Scrivere un programma C++ che chieda all'utente di riempire un array di 5 interi. Al termine dell'input, gli elementi dell'array devono essere messi al contrario.

Per esempio l'utente inserisce **`arr1={1,12,54,79,39}`** al termine del programma, deve risultare che **`arr1={39,79,54,12,1}`**

Serve una funzione **`swap`** che scambi due elementi (passaggio dei parametri per indirizzo)

Nel programma, lo scorrimento dell'array deve essere effettuato utilizzando l'aritmetica dei puntatori

71

Puntatori e strutture/classi/union

Esiste una sintassi semplificata per utilizzare i puntatori con dati di tipo aggregato. Per esempio, data la seguente struttura…

```
struct PUNTO {
      float x;
      float y;
};

PUNTO pnt, *pntPtr;
```

72

Puntatori e strutture/classi/union

In virtù delle diverse precedenze sono equivalenti:

```
pnt.x=0.0;
(*pntPtr).x=0.0;

pntPtr->x=0.0;
```

L'operatore *freccia* **->** semplifica di molto la sintassi dei puntatori specie quando le relative espressioni, riguardano più strutture innestate…

73

Gestione dinamica della memoria (prologo)

74

Gestione dinamica della memoria (prologo)

Il C++ consente ai programmatori di allocare e deallocare memoria a *runtime* per tutti i tipi di dato. Questa caratteristica è nota come *Gestione dinamica della memoria.*

Questa capacità del C++, permette di riservare spazio in memoria per nuovi dati che si sono resi necessari durante l'esecuzione e la cui dimensione non era nota al momento della scrittura del programma.

Allo scopo il C++ rende disponibili gli operatori **`new`** e **`delete`**

75

Gestione dinamica della memoria (prologo)

L'operatore new, crea un oggetto del tipo e della misura indicata. Possibilmente fornendo anche un valore di inizializzazione

```
int array[20], *dynArray, dsize
float *fp1;
…
fp1=new float (3.14);
cin >> dsize;
…
dynArray = new int[dsize];
…
```

L'operatore new restituisce il puntatore all'oggetto creato

76

Gestione dinamica della memoria (prologo)

Per l'allocazione di memoria ai nuovi oggetti, la new attinge a un area di memoria a disposizione del programma proprio per questo scopo: lo *heap.*

Lo *heap* non è infinito. E' buona norma liberare la memoria allocata quando non è più necessaria. Per questo c'è l'operatore delete

```
…
delete fp1;
…
delete [] dynArray;
…
```

77