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

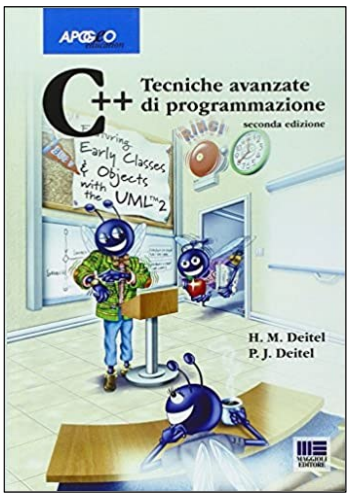
[FdP] H. M. Deitel, P. J. Deitel
C++ Fondamenti di programmazione
II ed. (2014) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8571-9

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

Le **`class`** in C++

6

Il puntatore **this**

Si «presenta» come un membro privato di tutte le classi definite in un programma C++:

- Non deve essere dichiarato e non è compreso nel computo (**sizeof**) della dimensione delle istanze della classe;
- Assume il tipo *puntatore alla classe* a cui è riferito;
- Contiene il puntatore all'oggetto in cui è utilizzato;

7

Esempio: *Il puntatore* ***this***

file: thiscavia.cpp

```
1  class cavia {
2          private:
3                  int val;
4          public:
5                  cavia() : val(0) {};
6                  int get() { return val; }
7                  cavia *set(int v) {
8                          val=v;
9                          return this;
10                 };
11                 cavia *set(cavia *c) {
12                         this->val=c->val;
13                         return this;
14                 }
15 …
```

8

Esempio: *Il puntatore* ***this***

file: thiscavia.cpp

```
6   …
7                   cavia *set(int v) {
8                           val=v;
9                           return this;
10                  };
…   …
```

this è di tipo **cavia ***

Definendo il metodo **set** in questo modo, nel codice possiamo fare cose come questa:

```
p=new cavia();
p->set(1)->set(2)->set(3);
```

che è del tutto equivalente a:

```
p->set(1);
p->set(2);
p->set(3);
```

9

Esempio: *Il puntatore* ***this***

file: thiscavia.cpp

```
1   class cavia {
2           private:
3                   int val;
4           public:
5                   cavia() : val(0) {};
6                   int get() { return val; }
7                   cavia *set(int v) {
8                           val=v;
9                           return this;
10                  };
11                  cavia *set(cavia *c) {
12                          this->val=c->val;
13                          return this;
14                  }
15  …
```

this è di tipo **cavia ***

Si riferisce all'attributo **val** dell'oggetto puntato da **c**

Si riferisce esplicitamente all'attributo **val** di *questo oggetto*

10

Esempio: *Il puntatore* **`this`**

file: thiscavia.cpp

```
16  …
27                  cavia *sum(cavia *c) {
28                          if(c!=this)
29                                  this->val+=c->val;
20                          return this;
21                  }
22  };
…   …
```

In un oggetto di classe **`cavia`**, le espressioni:

`val;` e **`this->val;`**

si riferiscono allo stesso membro, e sono entrambe corrette. Normalmente la seconda si utilizza per chiarezza o nella *ridefinizione degli operatori.*

`this` è utile quando non vogliamo che un certo metodo abbia per argomento lo stesso oggetto che l'ha invocato.

11

Template di funzioni: esempio

12

Esempio: *estrarre valori da un array*

Si scriva un programma in C++ che implementi due *funzioni template* per cercare alcuni valori dall' array **`src`** di tipo *arbitrario* **`T`**:

`void one_max(T *src, int srclen, T &max_el, int &pos)`

restituisce: in **`max_el`** l'elemento massimo dell'array puntato da **`src`** e lungo **`srclen;`** in **`pos`** la posizione dell'elemento massimo nell'array;

`void two_max(T *src, int srclen, T &max1, T &max2, int &pos1, int &pos2)`

restituisce: in **`max1`** e **`max2`** i due maggori elementi di **`src`** e in **`pos1`** e **`pos2`** le rispettive posizioni nell'array;

13

Esempio: *estrarre valori da un array*

`file: max_2max.cpp`

```
1  #include<iostream>
2  using namespace std;
3
4  template <class T>
5  void one_max(T *src, int srclen, T &max_elem, int &max_pos)
6  {
7          max_pos=0;
8          max_elem=src[0];
9
10         for (int i=1; i<srclen;i++)
11                 if (src[i]>max_elem){
12                         max_elem=src[i];
13                         max_pos=i;
14                 }
15 }
```

14

Esempio: *estrarre valori da un array*

file: max_2max.cpp

```
1  #include<iostream>
2  using namespace std;
3
4  template <class T>
5  void one_max(T *src, int srclen, T &max_elem, int &max_pos)
6  {
7          max_pos=0;
8          max_elem=src[0];
9
10         for (int i=1; i<srclen;i++)
11                 if (src[i]>max_elem){
12                         max_elem=src[i];
13                         max_pos=i;
14                 }
15 }
```

Il tipo «parametrizzato» è il tipo base dell'array

Ovviamente, l'elemento massimo estratto sarà dello stesso tipo

15

Esempio: *estrarre valori da un array*

file: max_2max.cpp

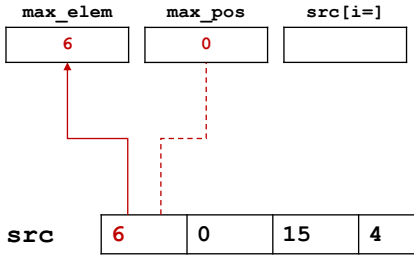
```
1  #include<iostream>
2  using namespace std;
3
4  template <class T>
5  void one_max(T *src, int srclen, T &max_elem, int &max_pos)
6  {
7          max_pos=0;
8          max_elem=src[0];
9
10         for (int i=1; i<srclen;i++)
11                 if (src[i]>max_elem){
12                         max_elem=src[i];
13                         max_pos=i;
14                 }
15 }
```

La funzione restituisce i valori cercati modificando direttamente i parametri passati, per riferimento, dal chiamante.

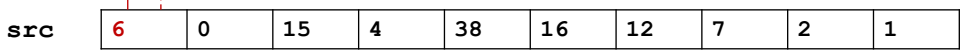
L'array, lo passiamo per indirizzo, e indichiamo anche la sua lunghezza.

16

Esempio: *estrarre valori da un array*



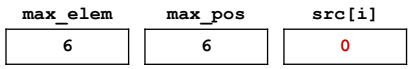

```
 7    max_pos=0;
 8    max_elem=src[0];
 9    for (int i=1; i<srclen;i++)
10         if (src[i]>max_elem){
11              max_elem=src[i];
12              max_pos=i;
13         }
```



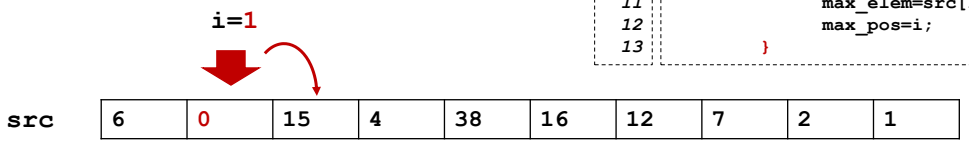
Inizialmente **`max_elem`** assume il valore di **`src[0]`** (il primo elemento dell'array); si scorre quindi l'array da sinistra a destra partendo dal secondo elemento **`i=1`**

17

Esempio: *estrarre valori da un array*



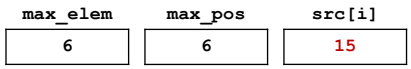

```
 7    max_pos=0;
 8    max_elem=src[0];
 9    for (int i=1; i<srclen;i++)
10         if (src[i]>max_elem){
11              max_elem=src[i];
12              max_pos=i;
13         }
```



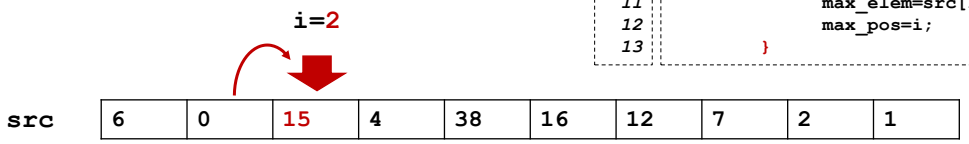
Si confrontano, di volta in volta, il valore dell'elemento in posizione *i*-esima **`src[i]`** con il valore corrente di **`max_elem`**: se l'esito del confronto è negativo si procede verso destra di una posizione (prossima iterazione del ciclo **`for`**)

18

Esempio: *estrarre valori da un array*



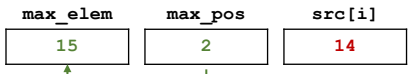

```
 7    max_pos=0;
 8    max_elem=src[0];
 9    for (int i=1; i<srclen;i++)
10         if (src[i]>max_elem){
11              max_elem=src[i];
12              max_pos=i;
13         }
```



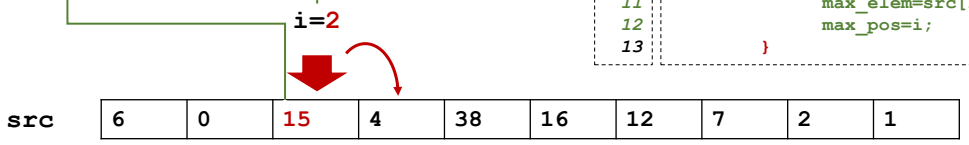
se l'esito del confronto tra **`src[i]`** e il valore corrente di **`max_elem`**: è positivo …

19

Esempio: *estrarre valori da un array*




```
 7    max_pos=0;
 8    max_elem=src[0];
 9    for (int i=1; i<srclen;i++)
10         if (src[i]>max_elem){
11              max_elem=src[i];
12              max_pos=i;
13         }
```



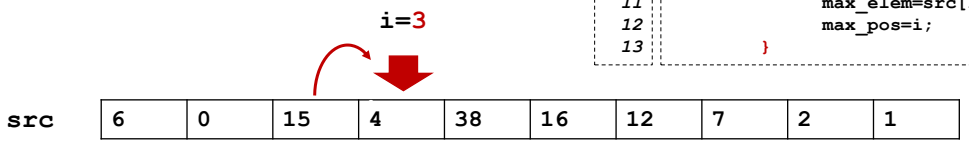
…il valore di **`src[i]`** si conserva in **`max_elem`**, la posizione i viene assegnata a **`max_pos`** e si procede verso destra (nuova iterazione del ciclo **`for`**), alla ricerca di nuovi eventuali massimi.

20

Esempio: *estrarre valori da un array*

max_elem	max_pos	src[i]
15	2	4

```
 7    max_pos=0;
 8    max_elem=src[0];
 9    for (int i=1; i<srclen;i++)
10         if (src[i]>max_elem){
11              max_elem=src[i];
12              max_pos=i;
13         }
```



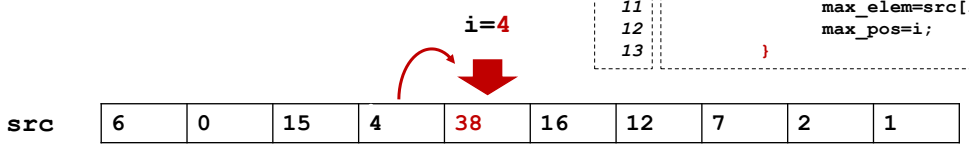
Si confrontano, di volta in volta, il valore dell'elemento in posizione i-esima **`src[i]`** con il valore corrente di **`max_elem`**: se l'esito del confronto è negativo si procede verso destra

21

Esempio: *estrarre valori da un array*

max_elem	max_pos	src[i]
15	2	38

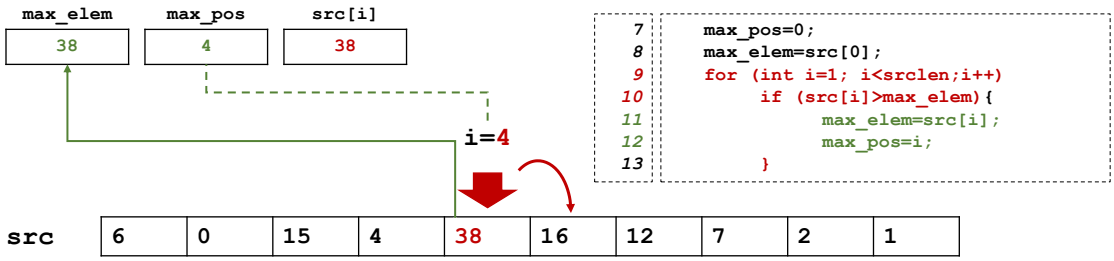
```
 7    max_pos=0;
 8    max_elem=src[0];
 9    for (int i=1; i<srclen;i++)
10         if (src[i]>max_elem){
11              max_elem=src[i];
12              max_pos=i;
13         }
```



se l'esito del confronto tra **`src[i]`** e il valore corrente di **`max_elem`**: è positivo …

22

Esempio: *estrarre valori da un array*

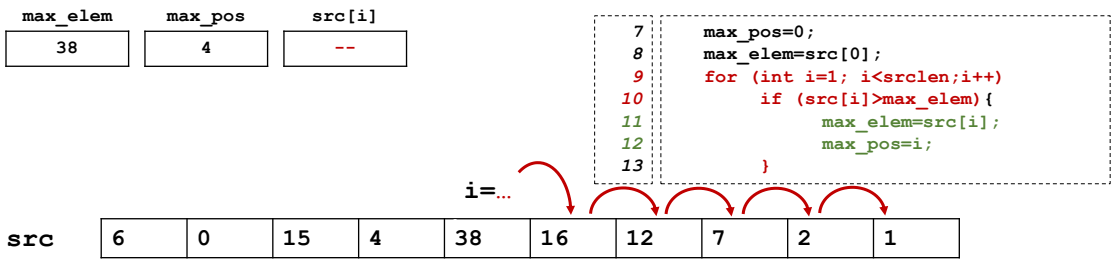



```
 7    max_pos=0;
 8    max_elem=src[0];
 9    for (int i=1; i<srclen;i++)
10         if (src[i]>max_elem){
11              max_elem=src[i];
12              max_pos=i;
13         }
```

…il valore di **`src[i]`** si conserva in **`max_elem`**, la posizione i viene assegnata a **`max_pos`** e si procede verso destra (nuova iterazione del ciclo **`for`**), alla ricerca di nuovi eventuali massimi.

23

Esempio: *estrarre valori da un array*




```
 7    max_pos=0;
 8    max_elem=src[0];
 9    for (int i=1; i<srclen;i++)
10         if (src[i]>max_elem){
11              max_elem=src[i];
12              max_pos=i;
13         }
```

…il valore di **`src[i]`** si conserva in **`max_elem`**, la posizione i viene assegnata a **`max_pos`** e si procede verso destra (nuova iterazione del ciclo **`for`**), alla ricerca di nuovi eventuali massimi.

24

Esempio: *estrarre valori da un array*

max_elem	max_pos	src[i]
38	4	--

```
 7    max_pos=0;
 8    max_elem=src[0];
 9    for (int i=1; i<srclen;i++)
10         if (src[i]>max_elem){
11              max_elem=src[i];
12              max_pos=i;
13         }
```

i=srclen

src	6	0	15	4	38	16	12	7	2	1

…terminato il **for** il chiamante ottiene i valori dell massimo elemento e la sua posizione nell'array nei parametri reali della funzione di cui **max_elem** e **max_pos** avevano «ricevuto» il riferimento.

25

Esempio: *estrarre valori da un array*

file: max_2max.cpp

```
 1  #include<iostream>
 2  using namespace std;
 3
 4  template <class T>
 5  void one_max(T *src, int srclen, T &max_elem, int &max_pos)
 6  {
 7          max_pos=0;
 8          max_elem=src[0];
 9
10          for (int i=1; i<srclen;i++)
11                  if (src[i]>max_elem){
12                          max_elem=src[i];
13                          max_pos=i;
14                  }
15  }
```

La funzione restituisce i valori cercati modificando direttamente i parametri passati, per riferimento, dal chiamante.

26

Esempio: *estrarre valori da un array*

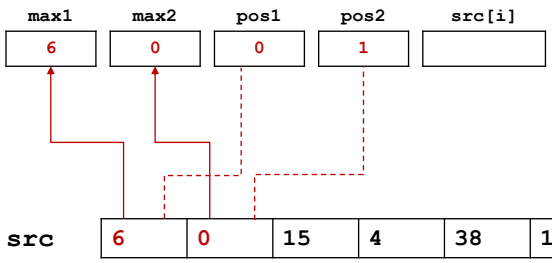
file: max_2max.cpp

```
…
17  template <class T>
18  void two_max(T *src, int srclen, T &max1, T &max2, int &pos1, int &pos2)
19  {
20          pos1=0;
21          pos2=1;
22          max1=src[pos1];
23          max2=src[pos2];
24
25          if (max1<max2){
26                  swap(max1,max2);
27                  swap(pos1,pos2);
28          }
…
```

La funzione restituisce i valori cercati modificando direttamente i parametri passati, per riferimento, dal chiamante.

27

Esempio: *estrarre valori da un array*




```
20          pos1=0;
21          pos2=1;
22          max1=src[pos1];
23          max2=src[pos2];
25          if (max1<max2){
26                  swap(max1,max2);
27                  swap(pos1,pos2);
28          }
```

Inizialmente **max1** e **max2** assumono i valore di **src[0]** e **src[1]** e **pos1** e **pos2** le rispettive posizioni (eventualmente se il secondo è maggiore, si scambiano) si scorre quindi l'array da sinistra a destra partendo dal terzo elemento **i=2**

28

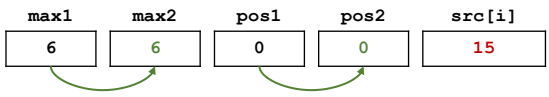
Esempio: *estrarre valori da un array*

file: max_2max.cpp

```
…
…
29          for(int i=2;i<srclen;i++)
30                  if(src[i]>max1){
31                          max2=max1;
32                          pos2=pos1;
33                          max1=src[i];
34                          pos1=i;
35                  } else if (src[i]>max2) {
36                          max2=src[i];
37                          pos2=i;
38                  }
39  }
…
…
```

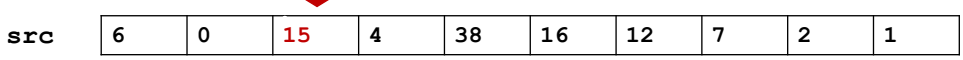
29

Esempio: *estrarre valori da un array*




```
29  for(int i=2;i<srclen;i++)
30     if(src[i]>max1){
31        max2=max1;
32        pos2=pos1;
33        max1=src[i];
34        pos1=i;
35     } else if (src[i]>max2) {
36        max2=src[i];
37        pos2=i; }     }
```

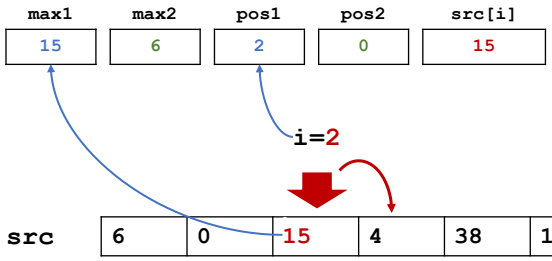
i=2



Confrontiamo **src[i]** prima con **max1**. Se dovesse risultare maggiore, allora **max2** e **pos2** prendono i vecchi valori di **max1** e **pos1**, quindi **max1** e **pos1** prendono i valori di **src[i]** e **i**. Si procede verso destra.

30

Esempio: *estrarre valori da un array*

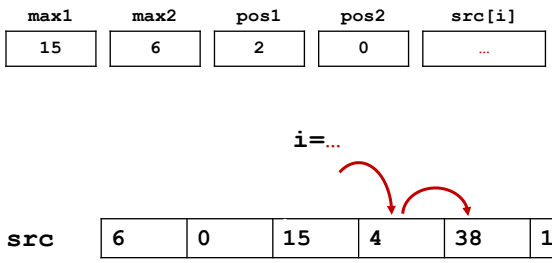



```
29  for(int i=2;i<srclen;i++)
30     if(src[i]>max1){
31        max2=max1;
32        pos2=pos1;
33        max1=src[i];
34        pos1=i;
35     } else if (src[i]>max2) {
36        max2=src[i];
37        pos2=i; }     }
```

Confrontiamo **`src[i]`** prima con **`max1`**. Se dovesse risultare maggiore, allora **`max2`** e **`pos2`** prendono i vecchi valori di **`max1`** e **`pos1`**, quindi **`max1`** e **`pos1`** prendono i valori di **`src[i]`** e **`i`**. Si procede verso destra.

31

Esempio: *estrarre valori da un array*

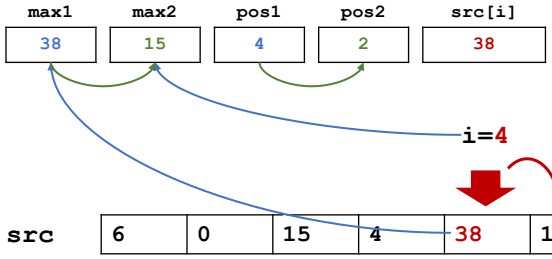



```
29  for(int i=2;i<srclen;i++)
30     if(src[i]>max1){
31        max2=max1;
32        pos2=pos1;
33        max1=src[i];
34        pos1=i;
35     } else if (src[i]>max2) {
36        max2=src[i];
37        pos2=i; }     }
```

Confrontiamo **`src[i]`** prima con **`max1`**. Se l'esito è negativo, si procede verso destra…

32

Esempio: *estrarre valori da un array*

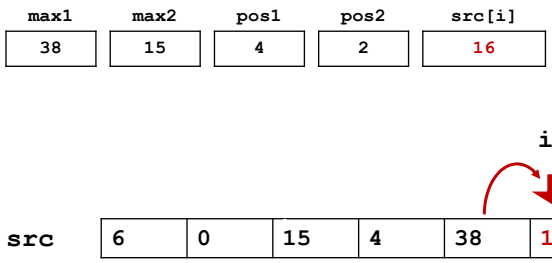



```
29  for(int i=2;i<srclen;i++)
30     if(src[i]>max1){
31        max2=max1;
32        pos2=pos1;
33        max1=src[i];
34        pos1=i;
35     } else if (src[i]>max2) {
36        max2=src[i];
37        pos2=i; }     }
```

Confrontiamo **`src[i]`** prima con **`max1`**. Se dovesse risultare maggiore, allora **`max2`** e **`pos2`** prendono i vecchi valori di **`max1`** e **`pos1`**, quindi **`max1`** e **`pos1`** prendono i valori di **`src[i]`** e **`i`**. Si procede verso destra.

33

Esempio: *estrarre valori da un array*

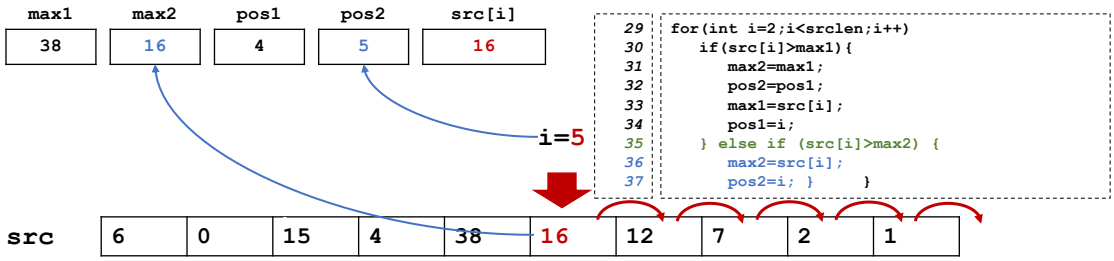



```
29  for(int i=2;i<srclen;i++)
30     if(src[i]>max1){
31        max2=max1;
32        pos2=pos1;
33        max1=src[i];
34        pos1=i;
35     } else if (src[i]>max2) {
36        max2=src[i];
37        pos2=i; }     }
```

Stavolta, **`src[i]`** non è maggiore di **`max1`** ma lo è rispetto a **`max2`** quindi, **`max1`** e **`pos1`** restano invariati e…

34

Esempio: *estrarre valori da un array*




```
29  for(int i=2;i<srclen;i++)
30     if(src[i]>max1){
31        max2=max1;
32        pos2=pos1;
33        max1=src[i];
34        pos1=i;
35     } else if (src[i]>max2) {
36        max2=src[i];
37        pos2=i; }      }
```

… **max2** e **pos2** prendono i valori di **src[i]** e **i**. Si procede verso destra, fino al termine…

35

Esempio: *estrarre valori da un array*

file: max_2max.cpp

```
41  template<class T>
42  void input (const char *messaggio, T *array, int len)
43  {
44          cout << messaggio <<endl;
45          for(int i=0;i<len;i++) {
46                  cout << "i="<<i<<": ";
47                  cin >> array[i];
48          }
49  }
50
51  template<class T>
52  void output (const char *messaggio, T* array, int len)
53  {
54          cout <<messaggio<<": { ";
55          for(int i=0;i<len;i++)
56                  cout << array[i] <<" ";
57          cout << "}"<<endl;
58  }
```

36

Esempio: *estrarre valori da un array*

file: max_2max.cpp

```
60  int main()
61  {
62          int interi[10],imax1,ipos1,fpos1,fpos2;
63          float reali[10],fmax1,fmax2;
64
65          input("Inserisci un array di 10 interi",interi,10);
66          one_max(interi,10,imax1,ipos1);
67          cout<<"L'elemento massimo e' : interi["<<ipos1<<"]="<<imax1<<endl;
68
69          input("Inserisci un array di 10 reali",reali,10);
70          two_max(reali,10,fmax1,fmax2,fpos1,fpos2);
71          cout<<"L'elemento massimo e' : reali["<<fpos1<<"]="<<fmax1<<endl;
72          cout<<"Il secondo massimo e' : reali["<<fpos2<<"]="<<fmax2<<endl;
73  }
```

37

Template di classi: esempio

38

Esempio: *la classe sequenza*

Si implementi una classe **`sequenza`** che contiene il puntatore a un array di tipo *arbitrario* **`TipoBase`**, e fornisca i seguenti metodi:

`sequenza(int len)`: costruttore di un oggetto sequenza di **`len`** elementi;

`TipoBase get(int i)`: restituisce: l'*i-esimo* elemento della sequenza;

`sequenza *set(int i, TipoBase elem)`: assegna **`elem`** alll'*i-esimo* elemento della sequenza. Restituisce il puntatore alla sequenza stessa;

`int max_pos(int i)`: restituisce la posizione dell'elemento massimo compreso tra l'*i-esimo* e l'ultimo elemento della sequenza;

`int min_pos(int i)`: restituisce la posizione dell'elemento minimo compreso tra l'*i-esimo* e l'ultimo elemento della sequenza;

39

Esempio: *la classe sequenza*

`sequenza *mischia(int seed)`: permuta casualmente gli elementi della sequenza. Utilizza il parametro per inizializzare il PRNG. Restituisce il puntatore alla sequenza stessa.

`sequenza *ordina_crescente()`: ordina gli elementi della sequenza in senso crescente, secondo la relazione d'ordine definita tra gli elementi di tipo **`TipoBase`**. Restituisce il puntatore alla sequenza stessa.

`sequenza *ordina_decrescente()`: ordina gli elementi della sequenza in senso decrescente, secondo la relazione d'ordine definita tra gli elementi di tipo **`TipoBase`**. Restituisce il puntatore alla sequenza stessa.

`bool ordinato_crescente()`: restituisce **`true`** se la sequenza risulta ordinata in senso crescente;

`bool ordinato_decrescente()`: restituisce **`true`** se la sequenza risulta ordinata in senso crescente;

`int size()`: restituisce la lunghezza della sequenza.

40

Esempio: *la classe sequenza*

file: sequenza.hpp

```
1  #ifndef _SEQUENZA_HPP_
2  #define _SEQUENZA_HPP_
3
4  #include <cstdlib>
5  using namespace std;
6
7  template <class TipoBase>
8  class sequenza {
9         private:
10                TipoBase *b;
11                int num;
12                sequenza *swap(TipoBase&, TipoBase&);
…  …
```

41

Esempio: *la classe sequenza*

file: sequenza.hpp

```
13        public:
14                sequenza();
15                sequenza(int);
16                sequenza *mischia(int);
17                sequenza *ordina_crescente();
18                sequenza *ordina_decrescente();
19                sequenza *set(int,TipoBase);
20                bool ordinato_crescente();
21                bool ordinato_decrescente();
22                TipoBase get(int);
23                int max_pos(int);
24                int min_pos(int);
25                int size();
26 };
…  …
```

42

Esempio: *la classe sequenza*

file: sequenza.hpp

```
27  template <class TipoBase>
28  sequenza<TipoBase>::sequenza(int n)
29  {
30          b = new TipoBase[n];
31          num = n;
32  }
33
34  template <class TipoBase>
35  sequenza<TipoBase> *sequenza<TipoBase>::swap(TipoBase &e1, TipoBase &e2)
36  {
37          TipoBase tmp;
38          tmp=e1;
39          e1=e2;
40          e2=tmp;
41          return this;
42  }
…   …
```

I metodi delle *classi template* devono essere definiti obbligatoriamente nello stesso file in cui è definita la classe stessa. Diversamente, la compilazione termina con un errore in *fase di linking*.

Esempio: *la classe sequenza*

file: sequenza.hpp

```
…   …
34  template <class TipoBase>
35  sequenza<TipoBase> *sequenza<TipoBase>::swap(TipoBase &e1, TipoBase &e2)
36  {
…   …
```

Tipo del risultato della funzione: *puntatore a un oggetto di classe* **sequenza**, definita utilizzando dati del tipo «generico» **TipoBase**

Identificatore della classe **sequenza** (con esplicitazione del tipo parametrizzato)

Prototipo della funzione.

Esempio: *la classe sequenza*

file: sequenza.hpp

```
27  template <class TipoBase>
28  sequenza<TipoBase>::sequenza(int n)
29  {
30          b = new TipoBase[n];
31          num = n;
32  }
33
34  template <class TipoBase>
35  sequenza<TipoBase> *sequenza<TipoBase>::swap(TipoBase &e1, TipoBase &e2)
36  {
37          TipoBase tmp;
38          tmp=e1;
39          e1=e2;
40          e2=tmp;
41          return this;
42  }
…   …
```

this è un attributo «speciale» di qualsiasi classe C++. E' utilizzato come un qualsiasi membro privato della classe ed è un puntatore all'istanza della classe stessa che lo utilizza.

45

Esempio: *la classe sequenza*

file: sequenza.hpp

```
43  template <class TipoBase>
44  sequenza<TipoBase> *sequenza<TipoBase>::mischia(int seed)
45  {
46          srand(seed);
47          int j=0;
48          for (int i=0;i<num;i++) {
49                  j=rand()%num;
50                  swap(b[i],b[j]);
51          }
52          return this;
53  }
…   …
```

Per ogni elemento della sequenza, ne estrae un altro a caso e li scambia di posto.

46

Esempio: *la classe sequenza*

file: sequenza.hpp

```
…
template <class TipoBase>
int sequenza<TipoBase>::max_pos(int st)
{
        int max_i=st;

        for(int i=st;i<num;i++)
                if (b[i]>b[max_i])
                        max_i=i;
        return max_i;
}
…
```

Restituisce la posizione dell'elemento massimo tra quelli della sequenza che sono compresi tra l'elemento in posizione **st** e l'ultimo elemento (**num-1**)della sequenza.

47

Intermezzo: *un algoritmo di ordinamento*

48

Un algoritmo di ordinamento

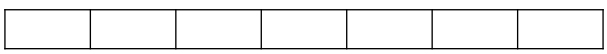
Dato l'array x di n numeri interi, descriviamo un algoritmo che ordini in maniera decrescente i suoi elementi:

49

Un algoritmo di ordinamento

Dato l'array x di n numeri interi, descriviamo un algoritmo che ordini in maniera decrescente i suoi elementi

Esempio: supponiamo che: $n = 7$

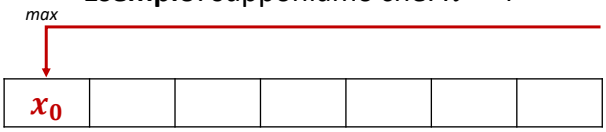


50

Un algoritmo di ordinamento

Dato l'array x di n numeri interi, descriviamo un algoritmo che ordini in maniera decrescente i suoi elementi

Esempio: supponiamo che: $n = 7$



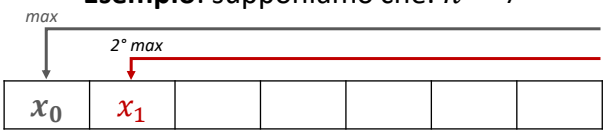
In un array ordinato in maniera decrescente, l'elemento più a sinistra: x_0 ha il massimo valore;

51

Un algoritmo di ordinamento

Dato l'array x di n numeri interi, descriviamo un algoritmo che ordini in maniera decrescente i suoi elementi

Esempio: supponiamo che: $n = 7$



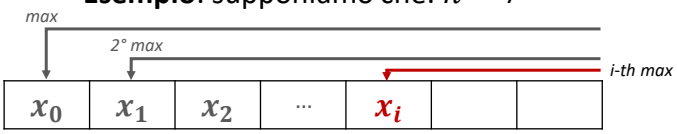
Se consideriamo la sequenza che va da: x_1 a x_{n-1}, il valore massimo è in x_1 e, inoltre $x_1 \leq x_0$

52

Un algoritmo di ordinamento

Dato l'array x di n numeri interi, descriviamo un algoritmo che ordini in maniera decrescente i suoi elementi

Esempio: supponiamo che: $n = 7$



Se consideriamo la sequenza che va da un generico x_i fino a x_{n-1}, il valore massimo è in x_i e inoltre, per ogni compreso j tra 0 e $i-1$ si ha che $x_i \leq x_j$

53

Un algoritmo di ordinamento

Dato l'array x di n numeri interi, descriviamo un algoritmo che ordini in maniera decrescente i suoi elementi

- Per i che va da 0 a n−1,
  - Cerchiamo, nella porzione di array compresa tra gli elementi i e $n-1$ l'elemento dal valore massimo
    - La posizione di questo massimo è assegnata alla variabile $imax$
  - Scambiamo di posto gli elementi x_i e x_{imax} in modo che adesso:
    - l'elemento massimo tra quelli conservati negli elementi di x che vanno da i a $n-1$, sia nella posizione i-esima;
    - Tutti gli elementi conservati nelle posizioni da $i+1$ a $n-1$ sono minori o uguali di x_i;

54

Un algoritmo di ordinamento

Dato l'array x di n numeri interi, descriviamo un algoritmo che ordini in maniera decrescente i suoi elementi

Esempio: supponiamo che $x = \{6,0,15,4,38,4,12\}$ e $n = 7$

6	0	15	4	38	4	12

Procediamo con la scelta l'elemento da porre nella posizione $i = 0$;

55

Un algoritmo di ordinamento

Dato l'array x di n numeri interi, descriviamo un algoritmo che ordini in maniera decrescente i suoi elementi

Esempio

6	0	15	4	38	4	12

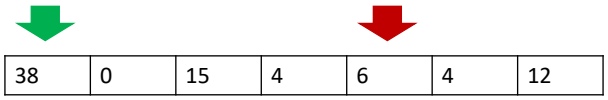
In primo luogo, cerchiamo l'elemento massimo tra le posizioni i e $n - 1$;

56

Un algoritmo di ordinamento

Dato l'array x di n numeri interi, descriviamo un algoritmo che ordini in maniera decrescente i suoi elementi

Esempio



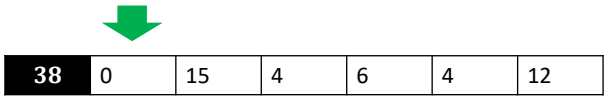
Quindi spostiamolo nella posizione $i = 0$ con uno scambio di posto;

57

Un algoritmo di ordinamento

Dato l'array x di n numeri interi, descriviamo un algoritmo che ordini in maniera decrescente i suoi elementi

Esempio



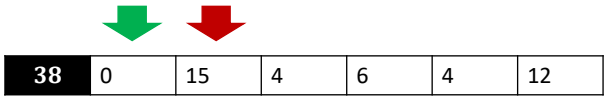
Abbiamo fissato l'elemento in posizione $i = 0$, andiamo avanti e dedicandoci al «sotto-array» costituito dagli elementi compresi tra 1 $(i \leftarrow i + 1)$ e $n - 1$;

58

Un algoritmo di ordinamento

Dato l'array x di n numeri interi, descriviamo un algoritmo che ordini in maniera decrescente i suoi elementi

Esempio



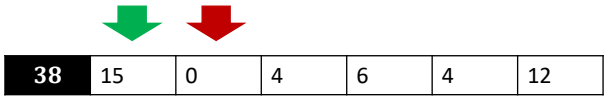
Cerchiamo ora l'elemento massimo

59

Un algoritmo di ordinamento

Dato l'array x di n numeri interi, descriviamo un algoritmo che ordini in maniera decrescente i suoi elementi

Esempio



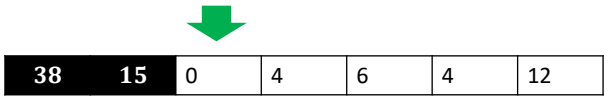
Effettuiamo lo scambio…

60

Un algoritmo di ordinamento

Dato l'array x di n numeri interi, descriviamo un algoritmo che ordini in maniera decrescente i suoi elementi

Esempio



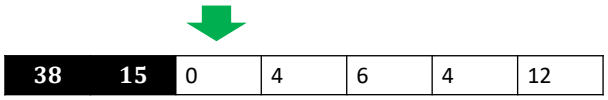
Andiamo avanti, ora $i = 2$ è la posizione da assegnare…

61

Un algoritmo di ordinamento

Dato l'array x di n numeri interi, descriviamo un algoritmo che ordini in maniera decrescente i suoi elementi

Esempio



Andiamo avanti, ora $i = 2$ è la posizione da assegnare…

L'algoritmo costruito in questo esempio è noto in letteratura con il nome di «*Selection Sort*»

62

Esempio: *la classe sequenza*

file: sequenza.hpp

```
…
54 template<class TipoBase>
55 sequenza<TipoBase> *sequenza<TipoBase>::ordina_decrescente()
56 {
57         int max_i;
58         for(int i=0;i<num;i++){
59                 max_i=max_pos(i);
60                 swap(b[i],b[max_i]);
61         }
62         return this;
63 }
…
```

Implementazione del *selection sort*

63

Esempio: *la classe sequenza*

file: sequenza.hpp

```
…
65 template<class TipoBase>
66 bool sequenza<TipoBase>::ordinato_decrescente()
67 {
68         for(int i=1;i<num;i++)
69                 if (b[i]>b[i-1])
70                         return 0;
71         return 1;
72 }
…
```

In una sequenza ordinata in maniera decrescente, ogni elemento è maggiore o uguale al successivo.

Se una coppia di elementi consecutivi non rispetta questa proprietà, la sequenza non è ordinata. Il metodo restituisce **`false`**

64

Esempio: *la classe sequenza*

file: provaseq.cpp

```
10  int main()
11  {
12          sequenza<int> dati(10);
13          int in;
14
15          cout << " *** Selection sort *** "<<endl;
16          cout << " Inserire 10 interi"<<endl;
17          for (int i=0;i<10;i++) {
18                  cout << "datir["<<i<<"]=";
19                  cin >> in;
20                 dati.set(i,in);
21          }
22          dati.mischia(23232);
23          show(dati);
24          if(!dati.ordinato_decrescente())
25                  cout<<"non è ordinato in maniera decrescente"<<endl;
26          dati.ordina_decrescente();
27          show(dati);
28          if(dati.ordinato_decrescente())
29                  cout<<"è ordinato in maniera decrescente"<<endl;
30  }
```

65