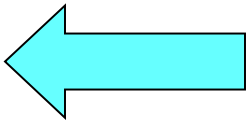
Lezione 10

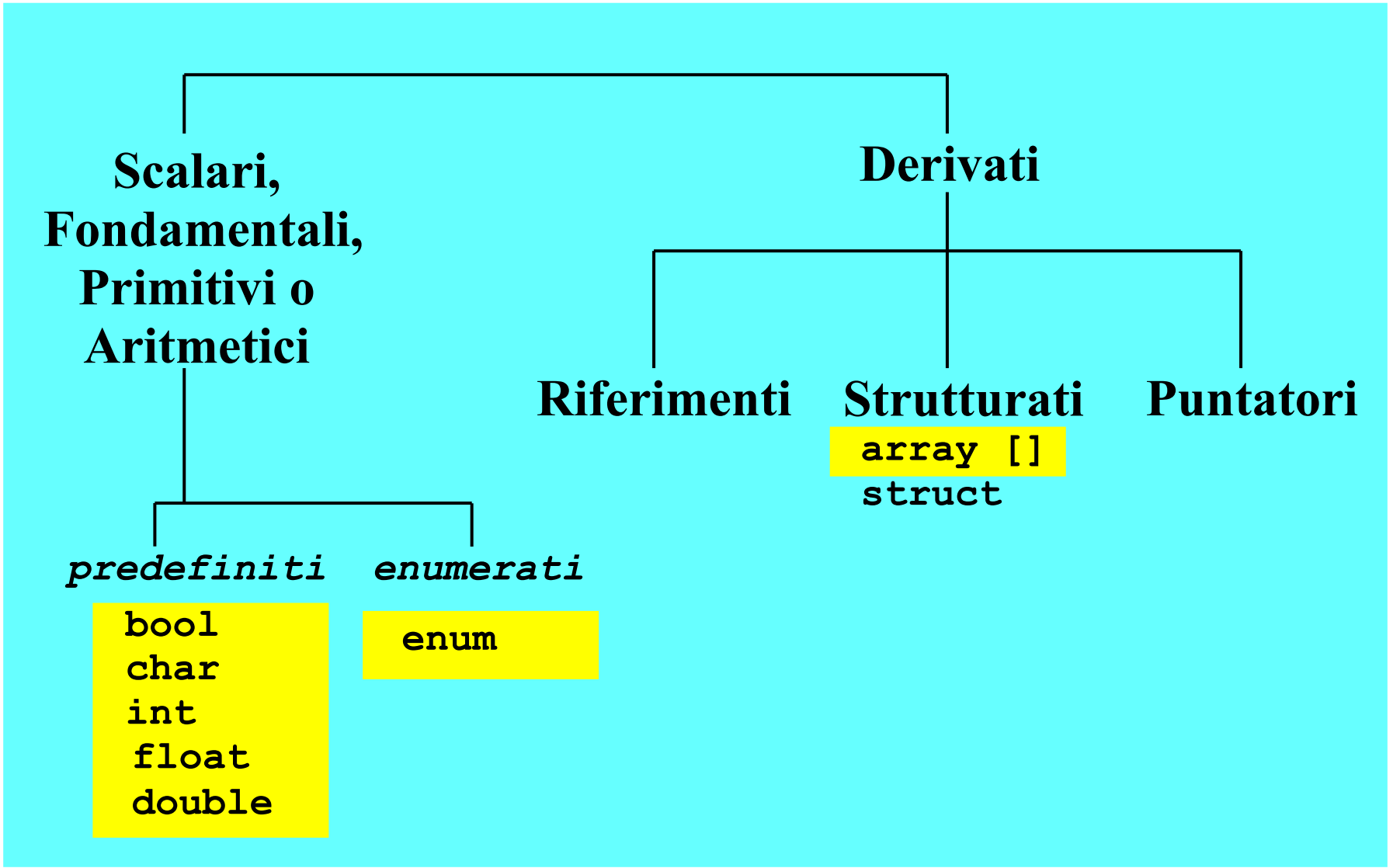
Array statici
Matrici statiche

Contenuto lezione

- Array statici
  - Passaggio alle funzioni
  - Vettori dinamici
  - Accesso fuori dall'array



Tipi di dato


Scalari, Fondamentali, Primitivi o Aritmetici
Derivati
Riferimenti
Strutturati
Puntatori
array []
struct
predefiniti
enumerati
bool
char
int
float
double
enum

Array

- Un *array* è una ennupla di *N* oggetti dello stesso tipo
  - Allocati in posizioni contigue in memoria

- **Selezione con indice**: ciascun elemento dell'array è denotato mediante:
  - nome dell'array
  - seguito da un indice intero compreso fra 0 e *N*-1, scritto tra parentesi quadre
- Esempio
  - Dato un array `A` di dimensione *N*
  - L'elemento *i-esimo* è denotato da `A[i]`, dove **0 <= i < *N***

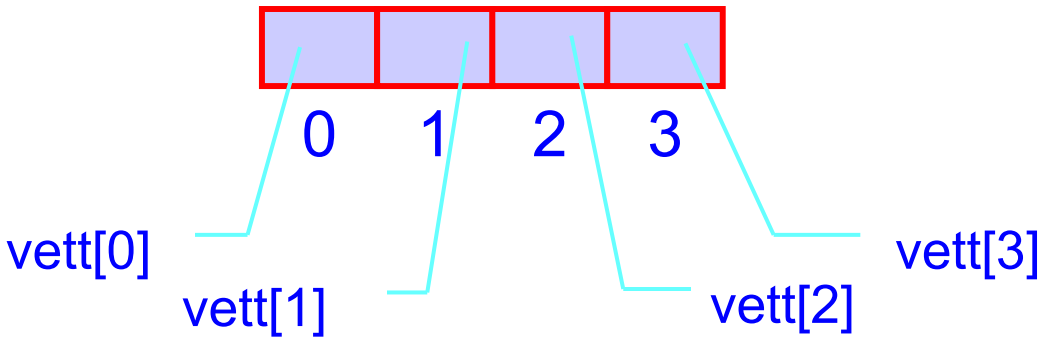
Definizione

- SINTASSI della definizione di una variabile o di una costante con nome di tipo **array statico**:

  [const] *<tipo_elementi_array>* *<identificatore>* **[***<espr-costante>***]** **;**

- Esempio: array statico di 4 elementi di tipo **int**

  int vett[4] ; // alloca spazio per 4 elementi **int** contigui



Dimensioni

Corretto

```
const int
      NUM_ELEM = 500;

int vett[NUM_ELEM];
```

Scorretto

```
int num_elem ;
cin>>num_elem ;

int vett[num_elem];
```

- La seconda forma è **fuori dallo standard**
  - Alcuni compilatori la consentono
- Il programma risultante non sarebbe più portabile

Intervallo indice

- Contrariamente ad altri linguaggi, il C/C++ non consente di scegliere il valore iniziale dell'indice di un array
  - Parte sempre da 0
  - Quindi, un array di *N* elementi ha sempre, necessariamente, indici da 0 a *N*-1 (inclusi)

Array e vettori

- Un array è un oggetto informatico che permette di implementare l'oggetto matematico vettore
  - Di fatto un array statico di *N* elementi corrisponde per definizione proprio ad un **vettore** statico **di *N* elementi**
  - Come vedremo a breve con un array statico si può però anche implementare un **vettore dinamico**, ossia un vettore con un numero di elementi che può variare durante l'esecuzione del programma
    - L'unico vincolo che abbiamo è che la dimensione massima raggiungibile dal vettore dinamico non può essere maggiore della dimensione dell'array statico utilizzato per rappresentarlo

Esercizi

- Svolgere i seguenti esercizi dell'ottava esercitazione:
  - *ins_stampa_array.cc*
  - *array_casuali.cc*

Assegnamento tra array

- Non è possibile assegnare il valore (contenuto) di un array ad un altro mediante un semplice assegnamento diretto
- Esempio:

```
int a[10], b[10] ;
…
a = b ; // SBAGLIATO !!!!!!!!!!!!!!
```

- L'unica soluzione è assegnare (copiare) gli elementi uno alla volta

Esercizio

- Dato un vettore di *N* interi, inizializzati da *stdin* o casualmente, si determini il valore massimo tra quelli memorizzati nel vettore e lo si stampi

Idea

- Assumi, come tentativo, che il “massimo momentaneo” sia il primo elemento del vettore
- Poi, confronta via via il “massimo momentaneo” con ciascuno dei successivi elementi del vettore
  - Ogni volta che trovi un elemento del vettore maggiore del “massimo momentaneo” sostituisci il “massimo momentaneo” con quell’elemento del vettore
- Dopo aver controllato tutti gli elementi del vettore, il “massimo momentaneo” corrisponderà al massimo del vettore

Verso un algoritmo

- Per trasformare la precedente idea in un algoritmo basta definire in modo preciso la struttura dati ed i passi da effettuare
- Come è fatta la struttura dati?

Struttura dati

- Array che realizza il vettore
- Costante **`N`** contenente la dimensione dell'array
- Variabile ausiliaria **`massimo`** destinata a contenere il massimo alla fine dell'algoritmo
- Variabile contatore ausiliaria per scandire l'array

Algoritmo

- Assegno alla variabile **`massimo`** il valore del primo elemento del vettore
- Poi, scandisco il vettore da 1 a **`N`**-1 confrontando **`massimo`** con ciascun elemento
- Se trovo un elemento del vettore maggiore di **`massimo`** sostituisco il valore di **`massimo`** con il valore di quell'elemento del vettore
- Dopo aver controllato tutti gli elementi del vettore, il massimo del vettore sarà contenuto nella variabile ausiliaria **`massimo`**

Programma

- *max_elem.cc* dell'ottava esercitazione

Variante per casa

- Dato un vettore di *N* interi, inizializzati da *stdin* o casualmente, si determini il valore massimo e si stampi sia il massimo sia la posizione del vettore in cui questi compare

Proposta struttura dati

- Array che realizza il vettore
- Costante **`N`** contenente la dimensione dell'array
- Variabile ausiliaria **`massimo`** destinata a contenere il massimo alla fine dell'algoritmo
- Variabile ausiliaria **`pos_massimo`** destinata a contenere la posizione dell'elemento di valore massimo alla fine dell'algoritmo
- Variabile contatore ausiliaria per scandire l'array

Programma

- *max_pos_elem.cc*
- Si può fare meglio?
  - Servono due variabili **`massimo`** e **`pos_massimo`**, o ne basta una?
  - *max_pos_elem2.cc*

Rischi di errore

- Dove è memorizzata implicitamente la dimensione di un array?
  - Da nessuna parte!
- Nel linguaggio C/C++ **non è previsto nessun controllo della correttezza degli indici** (inferiore e superiore) di un array
  - Per esempio, nel caso di dichiarazione:
    `int vettore[100];`
    istruzioni del tipo
    `vettore[105]=54;` `vettore[100]=32;`
    verrebbero accettate dal compilatore senza segnalazione di errori
- Tali errori possono causare fallimenti in modo impredicibile a tempo di esecuzione (per corruzione della memoria del programma)

Inizializzazione array 1/2

- Un array può essere inizializzato (solo) all'atto della sua definizione
- Notazione:

[const] *<tipo_elementi_array> <identificatore>* **[***<espr-costante>***] =**
{ *<espr1>, <espr2>, ..., <esprN>* **}**

- Esempi:

```
int a[3] = { 7, 3, 1 } ;
char cv[4] = { 't', 'A', '8', '$'} ;
```

Inizializzazione array 2/2

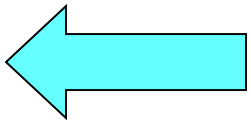
- Se un array è inizializzato, l'indicazione della dimensione si può omettere. In tal caso la dimensione è dedotta dal numero di valori inizializzati. I precedenti esempi sono equivalenti a:

```
int a[] = { 7, 3, 1 } ;
char cv[] = { 't', 'A', '8', '$'} ;
```

- Se invece le dimensioni sono indicate esplicitamente
  - Non si possono inizializzare più elementi di quanti se ne dichiara contenere l'array
  - Se se ne inizializzano meno, i restanti contengono il valore 0 o valori casuali a seconda che l'oggetto sia globale o locale
- Un array costante va inizializzato

Contenuto lezione

- Array statici
  - Passaggio alle funzioni
  - Vettori dinamici
  - Accesso fuori dall'array



Sintassi

<dichiarazione_parametro_formale_di_tipo_array> ::=
[const] *<tipo_elementi>* *<identificatore>***[ ]**

- Esempio definizione di una funzione con un parametro di tipo array:
  void fun(char c, int v[], …) { … }

- Esempio invocazione con passaggio di un array:

```
int A[4];
fun('b', A, ...);
```

- All'atto della chiamata si passa quindi semplicemente il nome dell'array

Tipologia passaggio

- Gli array sono **automaticamente passati per riferimento**
  - Se una funzione modifica l'array preso in ingresso (parametro formale), di fatto modifica l'array originario (parametro attuale)
  - Se si vuole evitare che questo accada, basta aggiungere il qualificatore **`const`** nella dichiarazione del parametro
    Esempio:

```
void fun(const int v[], …) ;
```

Dimensioni array

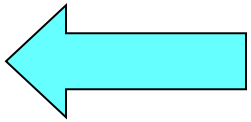
- Le informazioni sulle dimensioni dove sono?
  - Da nessuna parte!
- Come fa una funzione a sapere le dimensioni dell'array che le è stato passato?
  - Parametro addizionale
  - Variabile/costante globale
    - Soluzione potenzialmente più efficiente (permette di passare un parametro in meno) ma affetta da tutti i problemi degli oggetti globali precedentemente discussi

Esercizio

- Svolgere l'esercizio *calcola_somma.cc* dell'ottava esercitazione

Contenuto lezione

- Array statici
  - Passaggio alle funzioni
  - Vettori dinamici
  - Accesso fuori dall'array

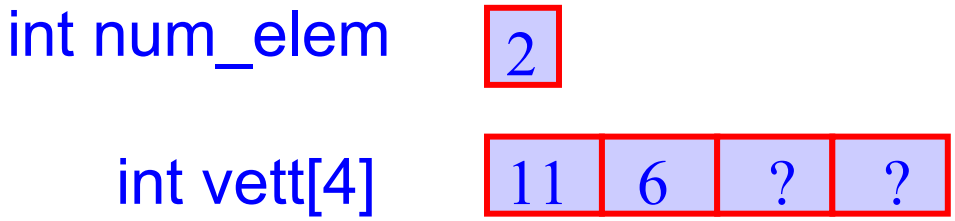


Vettori dinamici

- La dimensione *N* di un array statico non può essere determinata o modificata a tempo di esecuzione
- Al contrario, in molte applicazioni è necessario utilizzare vettori di dimensioni variabili o di dimensioni costanti ma note solo a tempo di esecuzione
- Uno dei modi più semplici per implementare un vettore di dimensioni variabili mediante un array è memorizzare il vettore in un array di dimensioni pari alla dimensione massima del vettore
- Questo vuol dire però che in determinati momenti dell'esecuzione del programma non tutte le celle dell'array saranno utilizzate

Implementazioni 1/2

- Come gestire l'occupazione parziale?
- Vi sono due tipiche soluzioni:
  - La prima è memorizzare il numero di elementi validi, ossia il numero di elementi del vettore dinamico, in una ulteriore variabile
- Esempio di implementazione di un vettore dinamico di al più 4 interi, e che ha al momento lunghezza 2

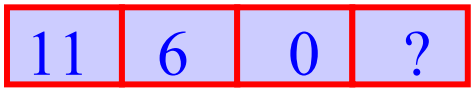


- I valori degli elementi dell'array successivi al secondo non hanno nessuna importanza, perché solo i primi due elementi sono validi

Implementazioni 2/2

- L'altra tipica soluzione è identificare il primo elemento non utilizzato con un valore che non appartiene all'insieme dei valori ammissibili per quel vettore. Tale elemento viene tipicamente detto **terminatore**. Ad esempio si può utilizzare
  - 0 per terminare un vettore di valori non nulli
  - -1 per terminare un vettore di valori positivi
- Esempio di implementazione di un vettore dinamico di al più 4 interi maggiori di zero, utilizzando il valore zero come terminatore

int vett[4]



- Il valore degli elementi dell'array successivi al terminatore non ha nessuna importanza, perché solo i primi due elementi sono validi

Oggetto astratto

- Le due soluzioni sono due esempi di implementazione di un oggetti astratto, il vettore dinamico, con uno o più oggetti concreti
  - nel primo caso un array più un contatore del numero di elementi validi
  - nel secondo caso un array
- Cogliamo l'occasione di questo esempio per evidenziare in modo concreto il processo di astrazione
  - l'oggetto astratto vettore dinamico astrae dai dettagli su come è implementato
  - sia che sia implementato con un array ed un contatore, o che sia implementato con solo un array, l'oggetto astratto vettore dinamico ha comunque le stesse identiche caratteristiche

Array e vettori astratti

- Riassumendo, mediante un array si può definire un oggetto astratto vettore, di cui si realizzano:
  - lunghezza variabile
    - mediante contatore numero di elementi validi o mediante elemento terminatore
  - assegnamento
    - mediante per esempio una funzione in cui si assegnano gli elementi uno ad uno
- Nella libreria standard di oggetti del C++ esiste anche l'oggetto astratto di tipo vettore (chiamato *vector*) che fornisce già queste e molte altre operazioni
  - Non useremo tale oggetto astratto in questo corso, ma implementeremo i vettori da noi, con le due tecniche appena viste

Esercizio

- Data una serie di rilevazioni di al più 100 temperature espresse in gradi Kelvin da memorizzare in un vettore, si calcoli la media delle temperature effettivamente fornite e si ristampino solo le temperature al di sopra della media
- Che valore hanno gli elementi dell'array non inizializzati?

Idee

- Chiedi in input il numero di valori *M* effettivamente rilevati, e leggi tutti questi valori
- Somma tutti i valori inseriti
- La media cercata è data dalla somma precedente divisa per *M*

Algoritmo

- Assumi che vi possano essere fino a 100 temperature, ma chiedi in input il numero di valori *M* effettivamente rilevati
- Leggi *M* valori non negativi e inseriscili in un vettore nelle posizioni da 0 a *M*-1
- Somma tutti gli *M* valori del vettore
- La media cercata è data dalla somma precedente suddivisa per *M*
- Per tutti gli elementi del vettore stampa solo quelli di valore maggiore della media

Struttura dati

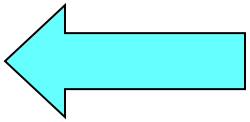
- Costante (int) per denotare la dimensione massima del vettore: **`max_M`**=100 (°)
- Array di reali di dimensione pari a **`max_M`**
- Variabile (int) per indicare il numero di valori di temperature effettivamente letti da input: **`M`** (**`0 < M <= max_M`**)
- Una variabile ausiliaria (int) da usare come contatore della scansione del vettore
- Infine, una variabile ausiliaria reale con il doppio ruolo di accumulatore delle somme parziali e contenitore della media

Programma

- *media_temp.cc*

Contenuto lezione

- Array statici
  - Passaggio alle funzioni
  - Vettori dinamici
  - Accesso fuori dall'array



Esercizio

- E se volessimo realizzarne una variante in cui l'utente non comunica neanche il numero *M* di temperature da memorizzare, ma bensì segnala la fine della lettura inserendo un valore negativo?
- Supponiamo inoltre di implementare il vettore utilizzando un valore terminatore
  - Attenzione a cosa bisogna fare nel caso in cui si inseriscono *max_M* elementi !!!

Proposta programma

```
main()
{
  const int max_M = 100 ; double vett_temper[max_M] ;

  for (int i = 0; i < max_M; i++) {
    cin>>vett_temper[i];
    if (vett_temper[i] < 0) {
     vett_temper[i] = -1 ; // inseriamo il terminatore
       break ;
    }
  } // nota: se inseriamo max_M valori non vi sarà alcun terminatore!

  double media = 0. ;
  // in uscita dal ciclo conterra' il num di valori letti
  int num_val_letti ;
  for (num_val_letti = 0; vett_temper[num_val_letti] >= 0; num_val_letti++)
    media += vett_temper[num_val_letti];

  media /= num_val_letti ;

  cout<<"Media: "<<media<<endl<<endl ;

  ...
}
```

IL PROGRAMMA E' CORRETTO?

Risposta

- Errore:
  - Non si controlla di essere sempre dentro l'array nella fase di calcolo della somma delle temperature
  - Se nell'array fossero stati inseriti *max_M* elementi, si finirebbe per leggere fuori dall'array
    - Errore logico
    - Possibile fallimento del programma, oppure lettura di valori casuali
      - Come mai fallimento o lettura valori casuali?
      - La risposta è nelle seguenti slide

Contenuto memoria 1/3

- Sappiamo già che la memoria di un programma è utilizzata per memorizzare le variabili e le costanti con nome definite nel programma (e quelle allocate dinamicamente, come vedremo nelle prossime lezioni)
- Ma non solo, la memoria contiene anche delle informazioni non direttamente manipolabili dal programmatore
  - Codice delle funzioni
  - Codice e strutture dati aggiuntivi, di supporto all'esecuzione del programma stesso
- In merito alle strutture dati aggiuntive, senza entrare nei dettagli consideriamo solo che, quando il compilatore traduce il programma in linguaggio macchina, non si limita a creare solo il codice macchina che esegue ciascuna delle istruzioni esplicitamente inserite nel codice sorgente

Contenuto memoria 2/3

- Il compilatore di fatto aggiunge codice ulteriore, di cui vedremo qualche dettaglio in alcune delle prossime lezioni, che effettua operazioni di supporto all'esecuzione del programma stesso (e che non sono e non possono essere esplicitamente inserite nel codice sorgente in base alle regole del linguaggio)
  - Inizializzare i parametri formali con i parametri attuali all'atto della chiamata di una funzione
  - Allocare spazio in memoria quando viene eseguita la definizione di una variabile/costante con nome
  - Deallocare spazio dalla memoria quando finisce il tempo di vita di una variabile/costante con nome
  - ...

Contenuto memoria 3/3

- Per realizzare alcune di queste operazioni il compilatore inserisce nella versione in linguaggio macchina del programma le strutture dati aggiuntive precedentemente menzionate
- In definitiva, se si accede a celle di memoria al di fuori di quelle dedicate ad oggetti correttamente definiti, si rischia di accedere a
  - Porzioni della memoria non ancora utilizzate per alcuno scopo, tipicamente contenenti valori casuali
  - Memoria occupata da altri oggetti definiti dal programmatore
  - Memoria occupata da codice o strutture dati aggiuntivi

Errore gestione della memoria

- In ogni caso, accedere al di fuori delle celle di memoria dedicate ad oggetti correttamente definiti è un **errore di gestione della memoria**
  - Se l'accesso avviene in lettura si leggono di fatto valori casuali
  - Se l'accesso avviene in scrittura si rischia la cosiddetta corruzione della memoria del programma, ossia la sovrascrittura del contenuto di altri oggetti definiti nel programma o delle strutture dati aggiuntive precedentemente menzionate
    - Se si corrompono le strutture dati aggiuntive il comportamento del programma diventa **impredicibile**

Diritti di accesso alla memoria

- I processori moderni permettono di suddividere la memoria in porzioni distinte e stabilire quali operazioni si possono/non possono effettuare su certe porzioni della memoria, nonché quando si ha diritto di effettuarle
  - Ad esempio, il codice di un programma viene tipicamente collocato in una porzione della memoria del programma stesso che è etichettata come non modificabile
  - Alcune porzioni della memoria gestite direttamente dal sistema operativo possono non essere accessibili, neanche in lettura, dal programma

Eccezione di accesso illegale

- Se un programma tenta di scrivere/leggere in una porzione della memoria in cui non ha il diritto di effettuare tale operazione viene tipicamente generata dal processore una eccezione hardware di accesso illegale alla memoria
  - Le eccezioni hardware sono un meccanismo del processore che fa interrompere l'esecuzione dell'istruzione in corso e saltare immediatamente all'esecuzione di codice speciale dedicato alla gestione dell'eccezione
  - Tipicamente il codice di gestione delle eccezioni hardware di accesso illegale alla memoria termina immediatamente il programma (che quindi fallisce)

Segmentation Fault

- In ambiente UNIX l'eccezione scatenata da un accesso illegale alla memoria viene tipicamente chiamata *Segmentation Fault*
- Come sappiamo, le eccezioni sono poi gestite da codice dedicato e deciso dal sistema operativo nella fase di avvio
- Il codice di gestione dell'eccezione *Segmentation Fault* tipicamente termina forzatamente il processo che ha generato l'eccezione

Accesso fuori da un array

- In conclusione, accedere fuori da un array è un errore sotto due punti di vista:
  - Errore logico
  - Errore di gestione della memoria
    - Corruzione della memoria nel caso di accesso in scrittura

Versione corretta

- Tornando all'esercizio, la versione corretta del programma è in *media_temp2.cc* nell'ottava esercitazione

Esercizio

- Dato un vettore di al più *max_M*=5 elementi interi non nulli, si copino in un altro vettore solo gli elementi compresi tra 10 e 500
- Al termine, si stampi il numero di valori copiati nel secondo vettore

Idee

- Ipotesi: implementiamo il vettore utilizzando il valore 0 come terminatore
- Bisognerà ovviamente scandire tutti gli elementi del primo vettore
  - Mentre si scandisce il vettore, si può copiare ogni valore accettabile nel nuovo vettore ed incrementare, per il secondo vettore, un contatore diverso da quello utilizzato per scandire il primo vettore
- Infine bisognerà stampare il valore finale del contatore relativo al secondo vettore

Struttura dati

- Costante (int) per denotare la dimensione massima dei due vettori: max_M
  - Probabilmente il secondo vettore avrà meno valori ammissibili del primo, ma perché il programma funzioni in tutti i casi, entrambi i vettori devono essere dimensionati per contenere fino a max_M elementi
- Servono due vettori di double, ciascuno di dimensione pari a max_M
- Servono, poi, due variabili ausiliarie (int) come contatore della scansione del primo vettore e come contatore dei valori effettivamente copiati

Proposta programma

```
main()
{
     const int max_M = 5 ;
     int vett_uno[max_M] = { 100, 3, 200, 0, 300 } ;
     int vett_due[max_M];

     int conta=0;
     for (int i=0; vett_uno[i] != 0 && i<max_M; i++)
       if (vett_uno[i]>=10  &&  vett_uno[i]<=500) {
           vett_due[conta]=vett_uno[i];
           conta++;
       }
     if (conta < max_M) // altrimenti scriviamo "fuori"
         vett_due[conta] = 0; // inseriamo il terminatore
     cout<<"Sono stati copiati "<<conta<<" elementi"<<endl;
}
```

COSA STAMPA?
COME MAI?
E' CORRETTO?

Versione corretta

- C'è un errore logico: si accede all'elemento *i*-esimo **prima** di controllare di non essere fuori dall'array
- Versione corretta:
  - *copia_in_intervallo.cc*

Esercizio

- Dall'ottava esercitazione:
  - *array_pari.cc*
  - tutti i successivi esercizi

Funzioni per casa 1/2

- Implementare le seguenti funzioni:

1) void genera (int v[], int N, int TOT);

Creazione di un vettore di interi riempito con un numero casuale da 0 a TOT, per N elementi
Riceve: V, N, TOT - Restituisce: niente

2) void leggiord (int v[], int N);

Lettura di un vettore di interi letto da tastiera, per N elementi, valutando che il vettore sia inserito ordinatamente (cioè un dato è rifiutato se minore di quello inserito nella posizione precedente)

Riceve: V, N - Restituisce: niente

3) int pos (int v[], int N, int E);

Ricerca sequenziale di un elemento E in un vettore V di N elementi

Riceve: V, N, E - Restituisce: posizione dell'elemento (-1 se non esiste)

4) int ins (int v[], int N, int DIM, int E);

Inserimento di un elemento E nella posizione corretta in un vettore V ordinato di N elementi con al massimo DIM elementi, slittando a destra gli elementi successivi alla posizione di inserimento.

Riceve: V, N, DIM, E - Restituisce: il numero di elementi finale (N+1) se l'elemento è stato inserito (cioè se N<DIM), N altrimenti

Funzioni per casa 2/2

5) int canc (int v[], int N, int E);
Cancellazione di un elemento E in un vettore V ordinato di N elementi (slittando a sinistra gli elementi successivi alla posizione di cancellazione)
Riceve: V, N, E - Restituisce: il numero di elementi finale (N-1) se l'elemento è stato trovato e cancellato, N se l'elemento non è stato trovato, 0 se il vettore è vuoto

6) void stampa (int v[], int N);
Stampa di un vettore V di N elementi
Riceve: V, N - Restituisce: niente

7) int ricbin (int v[], int N, int E);
Ricerca binaria di un elemento E in un vettore ordinato V di N elementi
Riceve: V, N, E - Restituisce: posizione dell'elemento (-1 se non esiste)
Prima di chiamarla si richiami la funzione ord per accertarsi prima che il vettore sia ordinato

8) int ord (int v[], int N);
Verifica che il vettore V sia ordinato
Riceve: V, N - Restituisce: 1 se v è ordinato, 0 altrimenti

9) void fusione(int v1[], int v2[], int v3[], int N);
Fonde i due vettori ordinati v1 e v2 di N elementi, nel vettore (vuoto) v3.
Riceve: V1, V2, N - Restituisce: nulla
Prima di chiamarla si richiami la funzione ord per accertarsi prima che il V1 e V2 sono ordinati . Attenzione che la dimensione di V3 deve essere il doppio di quella di V1 e V2.

Altri esercizi

1)Si scriva una funzione somma() che riceve come parametri 3 vettori v1, v2, v3 e la loro dimensione N. La funzione confronta il primo elemento di v1 e il primo elemento di v2 e copia il maggiore in v3 come primo elemento; confronta il secondo elemento di v1 e il secondo elemento di v2 e copia il maggiore in v3 come secondo elemento; ... e così via.
La funzione restituisce il numero di volte in cui un elemento di v1 è risultato maggiore dell'elemento di v2 con cui è stato confrontato.

Si scriva poi un programma che definisce tre vettori vett1, vett2, vett3, chiede a tastiera i valori dei due vettori vett1 e vett2, richiama la funzione sopra descritta e stampa il vettore vett3 risultante e il numero restituito dalla funzione.

2) Realizzare un funzione conta che riceve in ingresso un vettore V di interi ed un elemento E e restituisce quante volte E è ripetuto in V. Scrivere poi un main() che riempie un vettore leggendo dei valori da tastiera e fermandosi quando viene digitato il numero sentinella 999. Poi stampa a video il numero che ha il maggior numero di ripetizioni nel vettore. Esempio:

Input: 15 3 5 3 7 15 5 21 15 6 9 15 5 999
Output: "il numero più ripetuto è il 15 con 4 ripetizioni"

3) Scrivere una funzione contavolte che conta quante volte un elemento x è presente in un vettore v di n elementi. La funzione riceve come parametri x, v, n. Utilizzare contavolte dentro ad una seconda funzione creaunici, per costruire, a partire da un vettore v1, un secondo vettore v2 che contiene solo gli elementi unici di v1, cioè presenti una sola volta in v1. Esempio:

v1: 2 4 3 2 7 1 3 5 1 8 9
v2: 4 7 5 8 9