32. Gli array

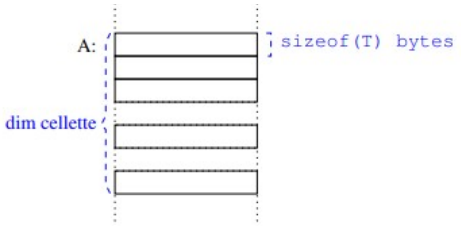
Array: sequenza finita di elementi consecutivi dello stesso tipo (*int, double, char, … , array di array, array di puntatori, …*).

Il numero di elementi di un array (dimensione) è fissata a priori. La dimensione può essere cambiata solo con array dichiarati dinamicamente.

Per un array di tipo T e dimensione dim, il compilatore alloca dim cellette consecutive di *sizeof(T) bytes* (allocazione statica): ovvero in caso di un array di interi di dimensione 8 vengono allocate 8 cellette di dimensione 4 bytes (la dimensione di un intero).

Un array rappresenta l'indirizzo di memoria del primo elemento della sequenza.

! Importante ! Con allocazione statica di array, tipicamente si definiscono array sufficientemente grandi, e poi se ne usa di volta in volta solo una parte. La dimensione virtuale di un array e la dimensione occupata realmente.



32.1 Definizione ed inizializzazione

Sintassi definizione

tipo id[dim];

tipo id[dim]={lista_valori};

tipo id[]={lista_valori};

Esempi

double a[25]; //array di 25 double

const int C=2;

*char b[2*c]={'a','e','i','o'};* // dimensione 4 char

d[]={'a','e','i','o','u'}; // dimensione 5

N.B.! Specificando un array in questo modo : *tipo id[dim];* si effettua una corretta definizione dell'array ma tutti i suoi elementi presentano valori casuali. Per questo è meglio scrivere *tipo id[dim]={}* in modo da inizializzare tutti gli elementi a 0 (anche se poi verranno magari inizializzati dall'utente).

! La dimensione dell'array deve essere valutabile al momento della compilazione:

- esplicitamente, tramite l'espressione costante dim (*tipo id[dim]={…]*)
- implicitamente, tramite la dimensione della lista di inizializzazione (*tipo id[]={…}*)

Se mancano elementi nella lista di inizializzazione, il corrispondente valore viene inizializzato allo zero del tipo T.

32.1.2 Operazioni non lecite sugli array

Sugli array non sono definite operazioni aritmetiche, di assegnamento o di input.

Le operazioni di confronto e di output sono definite, ma danno risultati imprevedibili: per tutte queste operazioni è necessario scrivere funzioni ad-hoc.

Esempi

int a[4] = {1,2}; //gli altri due valori sono inizializzati per default a zero

int b[4] = {1,2,3,4};

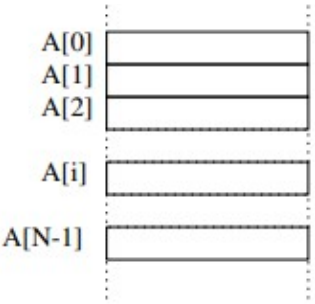
a++; a=b; cin >> a; //queste operazioni danno errori

cout << (a==b) << endl; cout << (a<=b) << endl; cout << a << endl; //queste operazioni permettono la compilazione ma sono imprevedibili (si stampa e compara l'*l-value*).

32.2 Operazioni sugli array: selezione con indice
L'unica operazione definita su un array è la selezione con indice (**subscripting**), ottenuta nella forma *id[expression]*, dove *id* è il nome di un array, mentre il *expression* (ovviamente può essere un'espressione variabile) è l'indice dell'elemento (è di tipo discreto convertibile in intero).

Importante! *identifier[expression]* è un'espressione dotata di indirizzo e può ricevere in input, essere assegnata, passata per riferimento, … .
*cin >> a[i]; a[n+3]=3*2; scambia(a[i], a[i+1])*

32.2.1 Range degli array
Gli indici degli elementi di un array di N dimensione vanno da 0 a N-1.
Esempio: se *i* vale 3, *a[i]=7* assegna il quarto elemento di *a* a 7.



32.2.2 Uscita dal range di un array
In C++, l'operatore "[]" permette di "uscire" dal range 0...dim-1 di un'array!
Ciò comporta effetti potenzialmente catastrofici in caso di errore: come ad esempio la sovrascrizione di un'altra variabile (*a[7]=8*, se la *dim* di *a è 7 (*cioè gli elementi vanno da 0 a 6*)* è un'operazione lecita ma va a lavorare sulla cella di memoria corrispondente a *a[7]* che non è una cella dell'array, ma quella di un'altra variabile definita nei dintorni).

È responsabilità del programmatore garantire che un operazione di subscripting non esca mai dal range di un'array! Cosa che purtroppo è discretamente probabile in operazioni con i cicli in cui si legge o si scrive un array.

Aiuto in compilazione: *g++ -fsanitize=bounds -fsanitize=bounds-strict* può aiutare ad identificare a run-time la maggior parte dei casi di errore di array subscripting out of bounds! Usare flags per assicurarsi di non uscire dal range. Con questo flag esce un errore di runtime quando si esce dal range dell'array.

Generalmente il C++ lascia le due celle di memoria prima e dopo quelle dell'array vuote, come celle cuscinetto per evitare questo tipo di errori. *g++ -fno-stack-protector* per rimuovere la protezione

32.3 Operazioni di assegnazione, cambio di valore e stampa di un array.

Assegnazione dei valori di un array in fase di input.
! La dimensione dell'array non può essere decisa dall'utente.

```
for (int i = 0; i < size; i++) {
    cin >> a[i];
 }
```

Stampa di un array

```
for (int i = size-1; i >= 0; i--){
  cout << a[i] << endl;  }
```

Per vedere elementi quanti elementi ha un array
cout << sizeof(array)/sizeof(T) << endl; //con *sizeof(array)* si ottiene il totale della memoria occupata dall'array (tanta quanti sono i suoi elementi, un elemento è come una variabile di tipo T)

Scambiare il valore di due elementi (utile per aggiungere un elemento in mezzo o invertire un array)

```
t=a[i];
a[i]=a[j];
a[j]=t;
```

Assegnazione di elementi all'array con il random
Chiaramente è necessario aver prima effettuato l'inizializzazione del seme.

```
void assegna(int a[], int dim){

  for(int i=0;i<dim;i++)
    a[i] = rand()%10+1;

  }
```

Queste operazioni possono essere fatte anche con delle semplici funzioni ricorsive.
Ad esempio, la stampa e la assegnazione di elementi random a un array con la ricorsiva. È possibile, anzi consigliabile, usare wrapper.

```
void assegna(int a[], int dim, int i){

  if(i<dim){
    a[i] = rand()%10+1;
    assegna(a,dim,++i);
  }

}
```

```
void stampa(int a[], int dim, int i){

  if(i<dim){
    cout << "a[" << i << "] = " << a[i] << endl;
    stampa(a,dim,++i);
  }

}
```

32.3.1 Altri esempi

Inserimento di un elemento in una posizione specifica
Importante: l'array ha una dimensione maggiore di uno rispetto a quanto viene inizializzato.

Versione iterativa

```
void inserimento (int a[], int d, int x, int i){
  for(int c=d; c>i; c--){
    a[c] = a[c-1];
  }
  a[i]=x;
  stampa(a, d+1);
}
```

Versione ricorsiva

```
void inserimento (int a[], int d, int x, int i){
  inserimento2(a,x,i,d);
  stampa(a, d+1); }

void inserimento2(int a[], int x, int i, int c){
  if(c>i){
    a[c]=a[c-1];
    inserimento2(a,x,i,--c);
  } else {
    a[i]=x;
  }
}
```

Eliminazione di un elemento

```
void eliminazione (int a[], int d, int x){
int count=0;
 for(int i=0;i<(d-count);i++){
    if(a[i]==x){
      count++;
      for(int c=i;c<d;c++){
           a[c]=a[c+1];
      }
      i=-1;
    }
  }
stampa(a, d-count);
 }
```

Concatenazione di un array

! Il terzo array che conterrà gli altri due deve essere di dimensione congrua a contenerli **!**

```
void concatena(int a[], int b[], int c[], int d){

  int ind=0;

  for(int i=0;i<d;i++){
    c[i]=a[i];
  }

  for(int i=d;i<2*d;i++,ind++){
    c[i]=b[ind];
  }

}
```

Inversione array

Usando un array di supporto:

```
void inverti(int a[], int b[], int d){
  for(int i=0;i<d;i++){
    b[i]=a[d-1-i];
  }
}
```

Usando lo stesso array:

```
void inverti(int a[], int d){
  for(int i=0;i<d/2;i++){
    int temp=a[i];
    a[i]=a[d-1-i];
    a[d-1-i]=temp;
  }
 }
```

32.4 Dimensione virtuale

Un modo per permettere all'utente di inserire il numero di elementi che vuole in un array è quello di usare una "dimensione virtuale" dell'array. Viene staticamente allocato un array molto grande (100 o più elementi, a seconda di ciò che dobbiamo fare) e poi quest'ultimo verrà riempito solo in parte.

È importante sottolineare come poi noi dovremmo usare come dimensione nei nostri algoritmi quella "virtuale", ovvero le celle che effettivamente contengono degli elementi. Quindi non la dimensione in fase di definizione, bensì quella esplicitata dall'utente.

Un modo è quello di chiedere proprio all'utente la dimensione dell'array, dopo averne allocato uno staticamente molto grande.

```
const int SIZE=100;
int a[SIZE]={};
int dim;
cout << "Quanti elementi vuoi inserire? " << endl;
cin >> dim;                    //ovviamente dim deve essere minore di SIZE (si possono usare dei
                               do-while di controllo)
```

Ora in TUTTI i nostri algoritmi assumeremo che la dimensione del nostro array sarà *dim*. Quindi quando dovremo effettuare l'assegnazione, o qualsiasi operazione con gli indici, l'indice andrà da 0 a *dim-1*.

Il secondo modo meno esplicito, consiste nel fare inserire all'utente i valori del nostro array e poi determinare di conseguenza la dimensione

```
const int SIZE=100;
int a[SIZE]={};
int num;
int i=0;

cout << "Inserisci dei numeri " << endl;
cout << "Premi un qualsiasi carattere non numerico per fermare l'inserimento" << endl;
while(cin << num){
a[i]=num;
i++;
}
cin.clear();
cin.ignore();
```

Non possiamo ricavare la dimensione del nostro array con questa operazione *int dim=sizeof(a)/sizeof(int);* perché *sizeof(a)* rappresenta la memoria occupata dalle 100 celle dell'array.

Quindi usiamo una variabile contatore (può semplicemente essere l'indice) nel ciclo *while* e poi definiamo *int dim=i;*

33. Funzioni con parametri di tipo array
Una funzione può avere un parametro formale del tipo “array di T”. Il corrispondente parametro attuale è un array di oggetti di tipo T
Esempio: *float sum(float v[],int n) {...}*
Tipicamente si omette la dimensione (*float v[]*, non *float v[dim]*) e si passa come secondo parametro la dimensione dell’array: *float sum(float v[], int dim).*
Tipicamente associato al numero di elementi effettivamente utilizzati (dimensione virtuale).

Dichiarazione: *T funz(T [], int);*
Definizione: *T funz(T a[], int DIM){};*
Chiamata: *funz(a, DIM);*

Passaggio di array: il contenuto dell’array non viene duplicato, evitando un possibile spreco di tempo, CPU e memoria.
Viene copiato solo l’indirizzo del primo elemento, in modo equivalente a passare gli elementi dell’array **per riferimento.** È possibile impedire di modificarli usando la parola chiave *const.*

! Gli array vengono passati per riferimento ! Quindi modificandoli nella funzione vengono modificati anche nel main.

33.1 Passaggio di parametri array costanti
È possibile definire passaggi di array in sola lettura (passaggio di array costante).
Sintassi: (*const tipo identificatore [], ...*)
Esempio: *int print(const int v[],...) {...}*

Chiaramente non è permesso modificare gli array.
Esempio: *v[5]=7;* //errore
Passaggio di informazione solo dalla chiamante alla chiamata: solo un input alla funzione.

Usato per passare array in input alla funzione:
- efficiente (no spreco di CPU e memoria)
- evita errori
- permette di individuare facilmente gli input della funzione

33.2 Operatore *sizeof*
Nel *main* l’istruzione *sizeof(a)*, restituisce il totale dei bytes occupato dall’array, che equivale al prodotto dei bytes del tipo degli elementi dell’array e il numero degli stessi.
Infatti facendo *sizeof(a)/sizeof(T)* otteniamo il numero di elementi dell’array, mentre facendo *sizeof(a)/DIM* otteniamo il numero di bytes occupati dal tipo.

In una funzione bisogna stare attenti, infatti *sizeof(arr) = sizeof(Type *)*, ovvero la dimensione del tipo puntatore (cioè 2 bytes).

33.4 Array e funzioni ricorsive
È frequente manipolare array con funzioni ricorsive. Uno degli argomenti delle funzioni (oltre all’array stesso e alla sua dimensione, virtuale o no) è una variabile di tipo *int* (inizializzata generalmente a 0 o a *DIM-1*) che funge da indice dell’array.

Tipicamente il parametro di ricorsione definisce il range dei sotto-array correntemente analizzati.

! Dividi et impera !