

IP QUIZ 2 E 3

lunedì 11 dicembre 2023 09:34

QUIZ 2

Consideriamo un array

```
int a[]={0,0,0,0};
```

indicare quali delle seguenti affermazioni sono vere (una o più di una)

- a. non sono in grado di giudicare la dimensione di a perche' non e' stata esplicitata
- b. la variabile a[0] è intera
- c. la variabile a (che in effetti non è variabile) contiene l'indirizzo base dell'array
- d. la variabile a è intera

la variabile a è un indirizzo

Gli array sono un tipo di dato strutturato che ci permette di descrivere sequenze di elementi *uniformi* (ossia dello stesso tipo).

Siamo in grado di accedere *direttamente* ad un elemento di un array con la sintassi a[i] dove i è un numero naturale da 1 a DIM (DIM = lunghezza dell'array stesso)

$\xrightarrow{\text{DIM}-1}$

Scegli una risposta:

- Vero
- Falso

Nell'utilizzare gli array, il problema dell'OUT OF BOUND

- a. Viene gestito automaticamente da funzionalità del linguaggio
- b. Ricade sul programmatore ✓
- c. Non so cosa sia l'out of bound

Tempo rimasto 0:33:07

Assumiamo che A e B siano array di 5 elementi interi

come posso creare una copia di A in B?

- a. for (unsigned int i=0;i<5;--i) B[i]=A[i];
- b. B=A;
- c. A=B;
- d. for (unsigned int i=0;i<5;++i) B[i]=A[i]; ✓
- e. non posso

Completare il frammento di codice **reverse** che copia da un array source ad un array dest gli elementi in ordine inverso (assumiamo che entrambi gli array abbiano lunghezza N)

```
for (int i=0; i<N;++)
    dest[i]=[1];
```

- a. sostituisci [1] con **source[N+1-i]**
- b. sostituisci [1] con **source[N-1-i]** ✓
- c. sostituisci [1] con **source[i]**
- d. sostituisci [1] con **source[N-i]**

Assumiamo di avere un array A di DIM elementi di tipo float, precedentemente inizializzato

Vogliamo calcolare la somma contenuto, quale soluzione e' errata?

- a. float sum=0; **for (unsigned int** i=DIM-1; i>=0; --i) sum-=A[i];
- b. float sum=0; **for (unsigned int** i=0; i<DIM; ++i) sum+=A[i];
- c. float sum=0; **for (unsigned int** i=0; i<DIM; ++i) sum=sum+A[i];
- d. float sum=0; **for (unsigned int** i=DIM-1; i>=0; --i) sum+=A[i];

Consideriamo l'algoritmo di ricerca binaria con la precondizione che la sequenza S di elementi considerata sia ordinata in maniera crescente

```
int first = 0;  
int last = DIM-1;  
int mid = (first+last)/2;  
bool trovato=false;  
  
while(first<=last && !trovato) {  
    mid = (first+last)/2;  
    if [1]  
        trovato=true;  
    else  
        if [2]  
            first = mid+1;  
        else  
            last = mid-1;  
}
```

Selezionare il completamento corretto per [1] e [2]

- a. [1] = ($S[mid]==elem$)
[2] = ($S[mid]>elem$)
- b. [1] = ($S[mid]<elem$)
[2] = ($S[mid]==elem$)
- c. [1] = ($S[mid]==elem$)
[2] = ($S[mid]<elem$)

✓

L'algoritmo selection sort permette di ordinare il contenuto di una sequenza di elementi.

Quali delle seguenti affermazioni sono vere?

- a. E' il miglior algoritmo di ordinamento presente in letteratura
- b. In ogni passo di ordinamento mette a posto un elemento. Inoltre in ogni passo abbiamo una sequenza che e' in parte ordinata e in parte no
- c. Ad ogni passo con ripetuti scambi (swap) di elementi porto l'elemento i-esimo nella posizione giusta
in ogni passo di ordinamento mette a posto un elemento
- d. E' un algoritmo di ordinamento "in place"

→ il Selection sort si basa sull'idea di minimizzare il numero di swap.
ogni elemento i-esimo viene aggiornato con un unico swap

Consideriamo la seguente definizione di tipo

```
struct Point {  
    double x; double y;  
};
```

L'espressione booleana (con **const double** TOLL=0.00001;)
(fabs(P1.x-P2.x)<TOLL)&&(fabs(P1.y-P2.y)<TOLL)

- a. e' corretta ma avrei potuto più semplicemente scrivere (fabs(P1-P2)<TOLL)
- b. può essere utilizzato per verificare se P1 e P2 sono uguali a meno di un valore piccolo
- c. E' sintatticamente errata
- d. Può essere utilizzata per verificare se P1 si trova in alto a sinistra rispetto a P2 sul piano cartesiano

Consideriamo

```
struct Date {  
    int year;  
    int month;  
    int day;  
};
```

Se dichiaro una variabile di tipo Date come segue

```
Date d={20,12,2022};
```

il compilatore segnalerà un errore sintattico

Scegli una risposta:

- Vero
- Falso

QUIZ 3

PUNTATORI L'operatore di deferenziazione * applicato ad una variabile puntatore p (*p) restituisce il valore della variabile puntata da p

Scegli un'alternativa:

- a. VERO
 b. FALSO

PUNTATORI L'operatore di referenziazione & applicato ad una variabile x (&x) restituisce il valore corrispondente all'indirizzo di x

Scegli un'alternativa:

- a. VERO
 b. FALSO

Consideriamo

int a=4;

int *p;

&p=a;

E' corretto?

Scegli una risposta:

- Vero
 Falso

Consideriamo il seguente frammento di codice

```
float * f1;  
float * f2;  
f1 = new float[5];  
// seguito da opportuna inizializzazione degli elementi a cui punta f1  
....  
f2 = f1;
```

Quali affermazioni sono corrette

- a. f1 e f2 puntano a porzioni consecutive in memoria dinamica, quindi usando l'aritmetica dei puntatori potrei passare da una all'altra
- b. f1 e f2 puntano a due diverse sequenze in memoria dinamica e potranno venir successivamente modificate in modo indipendente
- c. f2 è una copia profonda di f1
- d. f1 e f2 puntano alla stessa porzione di memoria dinamica
- e. f2 è una copia superficiale di f1

Qual è il contenuto dell'array v alla fine dell'esecuzione dei seguenti comandi?

```
int v[5]={3,6,9,12,15};
```

```
int *p=v;
```

```
p=p+1;
```

```
*p=4;
```

Scegli un'alternativa:

- a. 3 6 9 4 15
- b. 3 4 4 4 4
- c. 3 4 9 12 15
- d. 3 4 6 9 12 15

STRUTTURE DINAMICHE. Quale di queste affermazioni è vera?

Scegli un'alternativa:

- a. Le strutture dinamiche si trovano in una sezione della memoria primaria chiamata stack
- b. Le strutture dinamiche sono molto utili per gestire sequenze di elementi molto brevi
- c. L'area di memoria occupata dalle strutture dinamiche è allocata al tempo di esecuzione
- d. Le strutture dinamiche permettono di immagazzinare solo sequenze di elementi contigui
- e. Lo spazio occupato dalle strutture dinamiche è mantenuto in modo ordinato

Consideriamo un'implementazione di array dinamici come quella vista in classe, basata sulla seguente struct

```
struct dynamic_array {  
    int * store ;  
    unsigned int size ;  
};
```

Relativamente alla seguente funzione, quali affermazioni sono vere?

```
void una_funzione(dynamic_array &d, int index, int value) {  
    if ((index >= d.size) || (index < 0)) throw SOME_ERROR;  
    *(d.store+index)=value;  
}
```

- a. la funzione non e' in grado di gestire il caso in cui index sia out of bound
- b. la funzione permette di inserire il valore value nella posizione index dell'array
- c. la funzione permette di stampare il valore value nella posizione index dell'array
- d. la funzione restituisce valore speciale se index e' out of bound
- e. la funzione permette di verificare se il valore value si trova nella posizione index dell'array
- f. la funzione solleva un'eccezione se index e' out of bound

Consideriamo array dinamici relizzati come segue

```
struct my_vector{  
    int * store;  
    unsigned int size;  
    unsigned int capacity;  
};
```

La disponibilità di due campi distinti per size e capacity ci permette di (scegliere alternativa corretta)

- a. riservare uno spazio un po' piu' grande (capacity) di quanto non sia stato esplicitamente richiesto (size)
- b. definire la dimensione minima e la dimensione massima dell'array
- c. gestire un array con elementi non contigui
- d. nessuna delle altre opzioni

Qual è lo scopo della funzione qui di seguito?

```
vector<int> funzione(const vector<int> &v1, const vector<int> &v2){  
  
    vector<int> v(v1);  
    for (unsigned int i=0;i<v2.size();+ i)  
        v.push_back(v2.at(i));  
    return v;  
}
```

Scegli un'alternativa:

- a. restituire un vector unione di due vector
- b. restituire un vector concatenazione un vector di seguito ad un altro
- c. restituire un vector intersezione di due vector
- d. restituire un vector che crea una copia ribaltata di un altro vector
- e. restituire un vector che è la copia di un altro vector