Domande e Risposte - Wooclap

venerdì 2 ottobre 2020 14:28

Tipi interi

Qual è il risultato dell'operazione intera (3+2)/2.
Il risultato INTERO è 2 (con il resto di 1).

Gli unsigned int e gli int occupano spazi diversi di memoria.
Falso: lo spazio è lo stesso ma utilizzato in modo diverso.

I tipi long e short codificano interi in range di valori diversi.
Vero.

Prime istruzioni C++

Consideriamo il frammento di codice int num; num=num+1; Esso contiene un errore, di che tipo?

E' di tipo semantico, in quanto la sintassi del C++ è rispettata e il compilatore non da errori. Manca l'inizializzazione della variabile a cui sto facendo riferimento, mi viene quindi incrementato un valore non esatto.

Nell'assegnazione x = y/2; la variabile x è acceduta in scrittura, mentre y in lettura

Vero. Vi è una sovrascrittura del valore x, mentre il valore di y viene utilizzata in lettura nel calcolo.

Quale istruzione permette di acquisire da tastiera valori per le variabili x e y?

cin >> x >> y;

Espressioni Logiche

Supponiamo che int a=2, int b=4, int c=-1; qual è il risultato dell'espressione ((a!=b)&&(2!=a))

(a!=b) = true; (2!=a) = false; ((a!=b)**&&**(2!=a)) = false.

Supponiamo che int a=2, int b=4, int c=-1; qual è il risultato dell'espressione ((a>b)&&(a>c))||(a<=c)

(a>b) = false; (a>c) = false; (a<=c) = false; ((a>b)&&(a>c)) = false; ((a>b)&&(a>c))||(a<=c) = false;

Strutture di controllo

Il frammento di codice if ((Temp>=-20)||(Temp<=0)) cout << “Freddo dannato”; stampa“Freddo dannato” se la var Temp è compresa fra −20 e 0

Sbagliato. Occhio al significato dell'OR: basta che almeno una delle due condizioni sia rispettata (quindi anche 30, 50 sono considerate valide!).

if (minuti<=30) cout << “Sono le” << ora << “ e “ << minuti << endl; else cout << “Sono le” << ora<< “ meno “ << 60-minuti << endl; stampa ora e minuti adattando la frase a seconda che minuti sia minore-uguale o maggiore di 30

Sbagliato: nella seconda riga che stampa le ore sopra i 30 minuti, si passa all'ora successiva. Le 14.50 sono le "15 (ora+1)" meno "minuti".

if (f1==f2) cout << “sono uguali” << endl; date due variabili float f1 e f2, stampa “sono uguali” se sono uguali

E' semanticamente errato perche' non dà garanzie che l'uguaglianza sia verificata a causa di eventuali arrotondamenti dei float.

Array

Il problema dell'out of bound:

Non è gestito né segnalato dal linguaggio, ma va a tutto a discapito del programmatore verificare che non accada.

Consideriamo un array int a[]={0,0,0,0}; indicare quali delle seguenti affermazioni sono vere:

- la variabile a[0] è intera.
- la variabile a è intera. NO! FALSO. Essa è un identificatore che si accompagna all'indirizzo base di memoria, mentre tutti i suoi elementi (es. a[0]) sono variabili intere.
- la variabile a (che in effetti non è variabile) contiene l'indirizzo base dell'array.
- con l'istruzione sizeof(a) ottengo la lunghezza dell'array. FALSO.
  Sizeof è corretto, ma poiché faccio riferimento direttamente ad 'a' non restituisce correttamente la dimensione riservata ad ogni variabile intera. Dunque al momento non c'è modo per ottenere il valore della dimensione (lunghezza) dell'array se non portandoselo dietro.

Algoritmo di Ricerca, completa:

```
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;
}
```

Risposta:
[1] = (S[mid]==elem)
[2] = (S[mid]<elem)) - l'elemento si trova sulla destra, quindi porto il limite sinistro dell'intervallo al valore mid+1 (perché il valore all'indice mid ho già controllato che non fosse uguale a elem)

Struct

Dato il seguente codice:

```
La definizione di tipo

struct studente{
string nome;
string cognome;
int anno;
};

e la dichiarazione di variabile
studente S={"Rossi","Giulia", 1999};
```

Sono semanticamente errate inteso come 'scambio' di nome e cognome.
N.B. Non è un malfunzionamento, è un errore umano relativo alla propria conoscenza.
A livello sintattico è corretto inizializzare con le { } direttamente e in ordine i valori vengono assegnati ai campi interni alla struct seguendo dall'alto verso il basso.

Sempre in riferimento alla struct studente, come faccio ad accedere al primo campo (cognome)?

S.cognome

Dato il codice:

La variante

```
struct studente{
 string cognome;
 string nome;
data compleanno;
};
```

con data definita come segue

```
struct data{
 int anno;
 int mese;
 int giorno;
};
```

e' corretta?

Corretto.
Ricorda che in un codice completo bisogna dichiarare prima le struct che si vogliono utilizzare dopo, dunque la struct data sarebbe da dichiarare prima di studente se la si vuole utilizzare.

Quali istruzioni sono legali tra due variabili S1 ed S2 entrambe di tipo studente?

```
if (S1==S2) cout << "uguali\n";
```

```
if (S1<S2) cout << "ordinate\n";
```

S2=S1;

Ricorda che i confronti (==, <, >) si possono usare fra campi (elementi) di struct, non fra gruppi interi di struct. Posso invece assegnare (o inizializzare) una struct ad un'altra: ad

esempio a una struct vuota inserisco i valori nei campi di una struct di stesso tipo.

Funzioni

Qual è la differenza tra parametro formale e parametro attuale di una funzione

- Nessuna differenza, sono due nomi diversi per lo stesso concetto
- Nessuna sostanziale differenza, fanno riferimento alla stessa variabile (chiamata formale nella dichiarazione e attuale nella chiamata di una funzione)
- **Il parametro formale è una variabile dichiarata nell'intestazione della funzione, il parametro attuale è una variabile o un'espressione presente nella chiamata della funzione. Corretta!**
  N.B. Non fanno riferimento alla stessa variabile, anzi il parametro formale fornisce una variabile generale a cui più variabili diverse possono far riferimento all'atto della chiamata.

Consideriamo la funzione

```
int my_min(int n, int m) {
    if (n<m) return n;
    return m;
}
```

Quale delle seguenti istruzioni contenenti chiamate e' errata?

Waiting for next clap

- **sono tutte giuste**
- my_min(3,4);
- int a = my_min(4,5);
- d=my_min(1+1,4/5);
- int x=my_min(n,m); //assumiamo le var n e m esistano con valore nello scope della chiamata
- int x=my_min(a,b); //assumiamo le var a e b esistano con valore nello scope della chiamata

A livello sintattico e semantico sono tutte corrette (se poi utili o meno ("vanno a vuoto") non importa)

Puntatori

Quando dichiaro una variabile di tipo puntatore devo anche specificare il tipo base (ad esempio int *p è un puntatore a int) perche'

perche' in questo modo la variabile puntatore ha tutti gli elementi per accedere al contenuto della variabile puntata

e' un retaggio del C, oramai inutile

perche' la variabile puntatore di fatto e' del tipo di base

Nota bene: la variabile puntatore conserva un indirizzo di memoria della variabile puntata, devo conoscere quindi il tipo di quest'ultima. Per tale motivo si pone come stesso tipo della variabile puntata il tipo del puntatore. La variabile di tipo puntatore è di tipo PUNTATORE: la variabile puntatore si porta dietro un INDIRIZZO, non intero né altro, quindi non è del "tipo base della dichiarazione".

Consideriamo int x=10; int *p; quali delle seguenti istruzioni sono corrette

p=&x; *p=20;

*p=&x; p=20;

p=x; &x=20;

Nota bene: la variabile puntatore p punta a x (al suo indirizzo di memoria), con *p accedo al contenuto della variabile puntata (x) e aggiorno il suo contenuto con 20 (equivale a scrivere x=20).

Vector

Consideriamo la dichiarazione

vector<int> v;

quali affermazioni sono vere?

Waiting for next clap

- **v occupa spazio nello stack**
- v occupa spazio nello heap
- v ha una dimensione pari a una capacity superiore alla sua reale size
- **v.size() dovrebbe restituire 0**

Una "variabile con nome" qualsiasi essa sia, sta sempre nello stack. Poi essa può avere dei dati a cui puntare presenti nello heap, ma la variabile dal momento in cui dichiarata rimane nello stack. Quando vector è dichiarato vuoto occupa uno spazio stretto nello stack ma non vi è ancora presente una richiesta di incremento di memoria, dunque non vi è l'occupazione nello heap per ora.

Ricorsione

Gli elementi base (fondamentali) dell'algoritmo implementativo della ricorsione sono (programma fattoriale):

- If(n==0) return 1; (passo base)
- Return n*fattoriale(n-1); (passo induttivo)

Il meccanismo della ricorsione ci aiuta a realizzare soluzioni piu' efficienti

Falso.

N.B. non è sempre detto che ricorsione = efficienza!