Neuobičajeni tipovi podataka

Cerovina Stefana 1044/2014
Matematički Fakultet
Univerzitet u Beogradu

Sadržaj:

1. Strukture
2. Pokazivači
3. Globalni podaci

Uvod

1. Kada treba koristiti strukture umesto klasa
2. Prednosti i mane korišćenja pokazivača
3. Kako izbeći probleme pri radu sa globalnim podacima

Strukture

Strukture

- Strukture su složeni tipovi podataka, koji se sastoje od određenog broja (raznorodnih) elemenata
- Elementi strukture nazivaju se poljima strukture, i svako polje ima jedinstveni identifikator
- Polja strukture ne moraju biti istog tipa
- Polja strukture mogu biti proizvoljnog prostog ili složenog tipa
- Polju strukture se pristupa pomoću njegovog imena, koje je vidljivo samo unutar strukture kojoj polje pripada

Razlozi za korišćenje struktura(1)

- Jasno definisanje odnosa između podataka:
  - Primer (Visual Basic):

Bez korišćenja struktura

```
name = inputName
address = inputAddress
phone = inputPhone
title = inputTitle
department = inputDepartment
bonus = inputBonus
```

Sa korišćenjem struktura

```
employee.name = inputName
employee.address = inputAddress
employee.phone = inputPhone

supervisor.title = inputTitle
supervisor.department = inputDepartment
supervisor.bonus = inputBonus
```

Razlozi za korišćenje struktura(2)

- Korišćenje struktura radi uprošćavanja operacija nad blokovima podataka
  - Primer (Visual Basic): Zamena podataka između dva zaposlena

```
newName = oldName
newAddress = oldAddress
newPhone = oldPhone
newSsn = oldSsn
newGender = oldGender
newSalary = oldSalary
```

Razlozi za korišćenje struktura(3)

```
' swap new and old employee data
previousOldName = oldName
previousOldAddress = oldAddress
previousOldPhone = oldPhone
previousOldSsn = oldSsn
previousOldGender = oldGender
previousOldSalary = oldSalary

oldName = newName
oldAddress = newAddress
oldPhone = newPhone
oldSsn = newSsn
oldGender = newGender
oldSalary = newSalary

newName = previousOldName
newAddress = previousOldAddress
newPhone = previousOldPhone
newSsn = previousOldSsn
newGender = previousOldGender
newSalary = previousOldSalary
```

Razlozi za korišćenje struktura(4)

Pomoću strukture:

```
Structure Employee
   name As String
   address As String
   phone As String
   ssn As String
   gender As String
   salary As long
End Structure

Dim newEmployee As Employee
Dim oldEmployee As Employee
Dim previousOldEmployee As Employee
```

```
previousOldEmployee = oldEmployee
oldEmployee = newEmployee
newEmployee = previousOldEmployee
```

Razlozi za korišćenje struktura(5)

- Korišćenje struktura radi pojednostavljivanja liste parametara koja se prosleđuje funkciji
- Korišćenje struktura da bi se olakšalo održavanje

Pokazivači

Pokazivači(1)

- Korišćenje pokazivača je oblast koja je najsklonija greškama u modernom programiranju. Toliko je sklono greškama, da Java i Visual Basic nemaju pokazivački tip.

Pokazivači(2)

- Svaki pokazivač se sastoji iz dva dela:
  - memorijske adrese i
  - znanja kako da interpretira sadržaj te memorijske lokacije.
  - Kako ćemo interpetirati sadržaj neke memorijske adrese, zavisi od baznog tipa pokazivača.

Saveti pri radu sa pokazivačima(1)

- Izolovati operacije sa pokazivačima u klase ili funkcije(metode)
- Pisati funkcije(metode) za alokaciju pokazivača
- Deklarisati i definisati pokazivače u isto vreme
- Proveravati pokazivače pre korišćenja
- Proveravati promenljive na koju pokazivač pokazuje, pre korišćenja

Saveti pri korišćenju pokazivača(2)

- Koristiti dog-tag polja. Što češce proveravamo tag-field, to cemo biti bliži otkrivanju problema
- Alternativa korišćenju tag-fielda je korišćenje određenih polja dva puta. Ovo duplira kod samo na nekim mestima.

Saveti pri korišćenju pokazivača(3)

- Koristiti dodatne pokazivače radi jasnoće
  - Primer (C++):

Loše rešenje

```
void InsertLink(
   Node *currentNode,
   Node *insertNode
   ) {
   // insert "insertNode" after "currentNode"
   insertNode->next = currentNode->next;
   insertNode->previous = currentNode;
   if ( currentNode->next != NULL ) {
      currentNode->next->previous = insertNode;
   }
   currentNode->next = insertNode;
}
```

Saveti pri korišćenju pokazivača(4)

Bolje rešenje

```
void InsertLink(
   Node *startNode,
   Node *newMiddleNode
   ) {
   // insert "newMiddleNode" between "startNode" and "followingNode"
   Node *followingNode = startNode->next;
   newMiddleNode->next = followingNode;
   newMiddleNode->previous = startNode;
   if ( followingNode != NULL ) {
      followingNode->previous = newMiddleNode;
   }
   startNode->next = newMiddleNode;
}
```

Saveti pri korišćenju pokazivača(5)

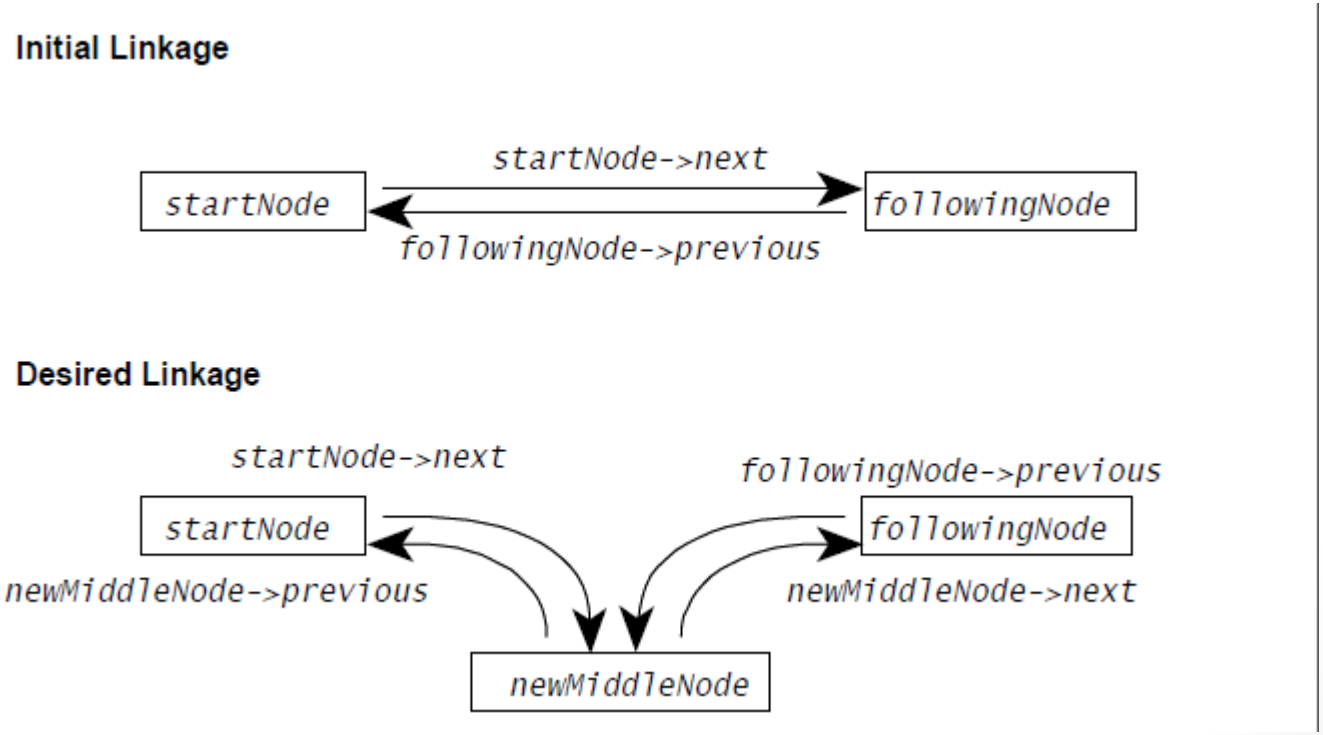
- Uprostiti komplikovane pokazivačke izraze

```
for ( rateIndex = 0; rateIndex < numRates; rateIndex++ ) {
   netRate[ rateIndex ] = baseRate[ rateIndex ] * rates->discounts->factors->net;
}
```

```
quantityDiscount = rates->discounts->factors->net;
for ( rateIndex = 0; rateIndex < numRates; rateIndex++ ) {
   netRate[ rateIndex ] = baseRate[ rateIndex ] * quantityDiscount;
}
```

Saveti pri korišćenju pokazivača(6)

- Crtati slike radi lakšeg razumevanja



Saveti pri korišćenju pokazivača(7)

- Oslobađati pokazivače u povezanoj listi u ispravnom redosledu
- Alocirati rezervnu memoriju
- Alocirati i dealocirati memoriju u istom bloku
- Pre oslobađanja, memoriju napuniti đubretom
  - Primer:

C++ Example of Forcing Deallocated Memory to Contain Junk Data

```
memset( pointer, GARBAGE_DATA, MemoryBlockSize( pointer ) );
delete pointer;
```

Saveti pri korišćenju pokazivača(8)

- Postaviti pokazivače na NULL posle oslobađanja ili brisanja
  - Primer:

C++ Example of Setting Pointers to NULL in a Replacement for *delete*

```
memset( pointer, GARBAGE_DATA, MemoryBlockSize( pointer ) );
delete pointer;
pointer = NULL;
```

Saveti pri korišćenju pokazivača(9)

- Proveravati vrednosti pokazivača pre brisanja
- Čuvati liste alociranih pokazivača
- Pisati funkcije za izbegavanje grešaka sa pokazivačima
  - Primer:

```
#define SAFE_DELETE( pointer ) { \
   ASSERT( pointer != NULL, "Attempting to delete NULL pointer."); \
   if ( IsPointerInList( pointer ) ) { \
      memset( pointer, GARBAGE_DATA, MemoryBlockSize( pointer ) ); \
      RemovePointerFromList( pointer ); \
      delete pointer; \
      pointer = NULL; \
   } \
   else { \
      ASSERT( FALSE, "Attempting to delete unallocated pointer." ); \
   } \
}
```

Saveti pri korišćenju pokazivača(10)

- Koristiti tehnike bez pokazivača ukoliko je to moguće

Pokazivači u programskom jeziku C++

- Razumevanje razlike između pokazivača(*) i reference(&)
- Osnovna razlika: referenca mora uvek referisati na objekat, dok pokazivač može pokazivati na NULL
- Ono na šta se odnosi referenca, ne može biti promenjeno kada je referenca već inicijalizovana

Pokazivači u programskom jeziku C++ - saveti(1)

- Koristiti pokazivače za prenošenje po adresi, a reference za prenošenje po vrednosti

C++ Example of Passing Parameters by Reference and by Value

```
void SomeRoutine(
   const LARGE_OBJECT &nonmodifiableObject,
   LARGE_OBJECT *modifiableObject
);
```

Pokazivači u programskom jeziku C++ - saveti(2)

- Pristupanje poljima pokazivačkog tipa vrši se pomoću: promenljiva->polje
- Pristupanje poljima referentnog tipa vrši se pomoću: promenljiva.polje
- Koristiti pametne pokazivače
  - Automatski oslobađa memoriju, sprečava curenje memorije

Pokazivači u programskom jeziku C - saveti(1)

- Koristiti eksplicitne tipove pokazivača radije nego podrazumevane. U tom slučaju nam kompajler daje upozorenja.
- Ako koristimo podrazumevane tipove pokazivača, moraćemo da kastujemo u određeni tip.
- Izbegavati kastovanje tipova
- Izbegavati smeštanja promenljive jednog tipa u prostor koji se odnosi na promenljivu drugog tipa
- Kastovanje isključuje mogućnost kompajlera da proveri da li se tipovi podudaraju

Pokazivači u programskom jeziku C - saveti(2)

- Poštovati pravilo zvezdice za prosleđivanje argumenata funkciji. Ako hoćemo da vratimo vrednost, mora stajati * ispred argumenta, u izrazu u kom joj se dodeljuje vrednost
- Koristiti sizeof za dobijanje veličine promenljive koja je alocirana

Globalni podaci

Globalni podaci

- Globalnim promenljivama se može pristupiti svuda u programu.
- Korišćenje je rizičnije od korišćenja lokalnih promenljivih

Uobičajeni problemi pri korišćenju globalnh promenljivih(1)

- Nepažljivo menjanje sadržaja globalne promenljive
- Dodeljivanje istoj promenljivoj dva ili više imena

Uobičajeni problemi pri korišćenju globalnh promenljivih(2)

Visual Basic Example of a Routine That's Ripe for an Aliasing Problem

```
Sub WriteGlobal( ByRef inputVar As Integer )
   inputVar = 0
   globalVar = inputVar + 5
   MsgBox( "Input Variable:  " & Str( inputVar ) )
   MsgBox( "Global Variable: " & Str( globalVar ) )
End Sub
```

Here's the code that calls the routine with the global variable as an argument:

Visual Basic Example of Calling the Routine with an Argument, Which Exposes Aliasing Problem

```
WriteGlobal( globalVar )
```

Since *inputVar* is initialized to *0* and *WriteGlobal()* adds *5* to *inputVar* to get *globalVar*, you'd expect *globalVar* to be 5 more than *inputVar*. But here's the surprising result:

The Result of the Aliasing Problem in Visual Basic

```
Input Variable:  5
Global Variable: 5
```

Uobičajeni problemi pri korišćenju globalnh promenljivih(3)

- Korišćenje jedne globalne promenljive od strane više niti
- Korišćenje dela koda jednog programa unutar drugog
- Redosled inicijalizacije podataka
- Globalni podaci oštećuju modularnost

Razlozi za korišćenje globalnih promenljivih

- Čuvanje globalnih vrednosti
- Imitacija imenovanih konstanti
- Imitacija enumerisanih tipova
- Pojednostavljivanje upotrebe zajedničkih podataka

Globalni podaci-saveti

- Koristiti globalne podatke samo kao krajnje rešenje
- Na početku deklarisati sve promenljive kao lokalne.
- Napraviti razliku između globalnih i klasnih podataka
- Koristiti funkcije za pristup podacima umesto globalnih podataka

Funkcije za pristup podacima-prednosti

- Svi pristupi promenljivoj nalaze se u funkcijama za pristup podacima
- Centralizovana kontrola podataka
- Prednosti sakrivanja informacija
- Funkcije za pristup se lako konvertuju u apstraktni tip

Kako koristiti funkcije za pristup podacima?(1)

- Sakriti podatke u klasi.
- Deklarisati podatke kao static (ili ekvivalentna ključna reč) radi osiguravanja da postoji samo jedna instanca tog podatka.
- Napisati funkcije koje obezbeđuju pristup i menjanje tih podataka.
- Zahtevati da kod izvan klase pristupa podacima samo preko funkcija za pristup.

Kako koristiti funkcije za pristup podacima?(2)

- Šta ako programski jezik ne podržava klase?
- Zahtevati da globalne promenljive počinju prefiskom g_
- Zahtevati da je pristup globalnim promenljivama omogućen samo unutar funkcije za pristup
- Ne stavljati na gomilu sve globalne podatke i njihove pristupne funkcije
- Koristiti zaključavanje za kontrolisanje pristupa globalnim podacima

Kako koristiti funkcije za pristup podacima?(3)

- Apstrahovati funkcije za pristup podacima

Direct Use of Global Data	Use of Global Data Through Access Routines
node= node.next	*account = NextAccount(account)*
node = node.next	*employee = NextEmployee(employee)*
node = node.next	*rateLevel = NextRateLevel(rateLevel)*
event = eventQueue[queueFront]	*event = HighestPriorEvent()*
event = eventQueue[queueBack]	*event = LowestPriorityEvent()*

Kako koristiti funkcije za pristup podacima?(3)

- Isti nivo apstrakcije za sve funkcije za pristup određenom podatku

Kako smanjiti rizik pri korišćenju globalnih podataka?(1)

- Definisati pravila imenovanja globalnih promenljivih, tako da se razlikuju od ostalih tipova promenljivih
- Napraviti listu globalnih promenljivih i svakoj pridružiti određeni opis
- Ne smeštati privremene rezultate u globalnu promenljivu. Ukoliko je potrebno promeniti vrednost globalne promenljive, uraditi to na kraju izračunavanja.

Kako smanjiti rizik pri korišćenju globalnih podataka?(2)

- Ne izbegavati korišćenje globalnih promenljivih grupisanjem svih podataka u jedan veliki objekat

Zaključak(1):

- Strukture mogu učiniti kod manje komplikovanim, lakšim za razumevanje i održavanje
- Kad god razmatramo korišćenje strukture, trebalo bi razmotriti da li bi bilo bolje napraviti klasu
- Rad sa pokazivačima je sklon greškama.Treba se zaštiti korišćenjem funkcija za pristup podacima ili klasa.

Zaključak(2):

- Izbegavati korišćenje globalnih podataka
- Ukoliko je neophodno korišćenje globalnih podataka, koristiti funkcije za pristup podacima

Kraj!

Hvala na pažnji!

Stefana Cerovina 1044/2014