Χρήσιμες έννοιες στη C++ και μερικές δομές δεδομένων

#19

Τμήμα Πληροφορικής και Τηλεπικοινωνιών (Άρτα)

Πανεπιστήμιο Ιωαννίνων

Γκόγκος Χρήστος

V 1.0

Προγράμματα C++ με ορίσματα γραμμής εντολών

- `int main(int argc, char* argv[]){…}`
  - `argc` είναι το πλήθος των ορισμάτων γραμμής εντολών, τα ορίσματα διαχωρίζονται με κενά μεταξύ τους.
  - `argv` είναι ένας πίνακας δεικτών προς λεκτικά που περιέχουν τα ορίσματα γραμμής εντολών

https://github.com/chgogos/oop/blob/master/various/COP3330/lect19/sample1.cpp

Αρχεία κεφαλίδων (header files) C++

- Τα αρχεία πηγαίου κώδικα χρησιμοποιούν την οδηγία `#include` προκειμένου να συμπεριλάβουν τα αρχεία κεφαλίδων.
- Μερικές φορές η χρήση αρχείων κεφαλίδων μπορεί να δημιουργήσει προβλήματα.
  - Κάνοντας include το ίδιο header file δύο ή περισσότερες φορές μπορεί να δημιουργηθούν λάθη «διπλής δήλωσης».
  - Κάνοντας include δύο φορές το `stdio.h` ωστόσο δεν δημιουργείται αντίστοιχο πρόβλημα. Γιατί;

```
$ g++ sample2.cpp
In file included from myheader2.h:1,
                 from sample2.cpp:5:
myheader1.h:1:7: error: redefinition of 'class
abc'
 class abc {
       ^~~
In file included from sample2.cpp:4:
myheader1.h:1:7: note: previous definition of
'class abc'
 class abc {
       ^~~
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect19/sample2.cpp

https://github.com/chgogos/oop/blob/master/various/COP3330/lect19/myheader1.h

https://github.com/chgogos/oop/blob/master/various/COP3330/lect19/myheader2.h

Header files στη C++

- Ο ακόλουθος μηχανισμός αποτρέπει το να συμπεριληφθεί το σώμα ενός header αρχείου πολλαπλές φορές.

```
#ifndef MYHEADER
#define MYHEADER
…
/* το σώμα του header
αρχείου */
#endif
```

ή

```
#pragma once
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect19/sample3.cpp

Μακροεντολές με παραμέτρους

- Οι μακροεντολές με παραμέτρους μοιάζουν και λειτουργούν παρόμοια με τις συναρτήσεις
  - `#define max(a,b) (a>b)?a:b`

- Οι μακροεντολές με παραμέτρους πρέπει να ορίζονται προσεκτικά αλλιώς μπορεί να παράγονται μη αναμενόμενα αποτελέσματα.
  - Ποιο είναι το πρόβλημα στην ακόλουθη μακροεντολή;

```
#define sum(a,b) a+b
…
cout << 10 * sum(2, 3) << endl; // αποτέλεσμα 23 και όχι 50 !
```

C++ λειτουργίες ανά bit (bitwise)

- 1 byte = 8 bits
- Ένα byte είναι η μικρότερη ποσότητα μνήμης η οποία μπορεί να αντιστοιχιστεί σε μια μεταβλητή (`char c`).
- Ωστόσο, μπορούμε να χρησιμοποιούμε κάθε επιμέρους bit με τους bitwise τελεστές:
  - bitwise and (ΚΑΙ): &
    - `0xFF & 0x00 == 0x00`
  - bitwise or (Ή): |
    - `0xFF & 0x00 == 0xFF`
  - bitwise exclusive or (ΑΠΟΚΛΕΙΣΤΙΚΟ Ή): ^
    - `0xFF & 0x00 == 0xFF`
  - left shift (ΑΡΙΣΤΕΡΗ ΟΛΙΣΘΗΣΗ): <<
  - right shift (ΔΕΞΙΑ ΟΛΙΣΘΗΣΗ): >>
  - bitwise complement (ΣΥΜΠΛΗΡΩΜΑ): ~
    - `~0xF0 == 0x0F`

Bitwise λειτουργίες

- `unsigned char x = 8` `(00001000)`
- `unsigned char x = 192` `(11000000)`

- `x = 00001000` `= 00001000`
- `y = 11000000` `= 11000000`
- `x&y = 00000000` `x|y = 11001000`

- Έλεγχος αν το δεύτερο bit του x είναι 1:
  - `if (x & 0x04 !=0)`
- Έτσι επιτυγχάνεται η σύνταξη λογικών εκφράσεων με βάση τα bits μιας μεταβλητής

Δομές δεδομένων

- Οι δομές δεδομένων επιτρέπουν την ευκολότερη συγγραφή προγραμμάτων.
- Οι δομές δεδομένων εστιάζουν στην οργάνωση των δεδομένων με κατάλληλο τρόπο έτσι ώστε να πραγματοποιούνται αποδοτικότερα λειτουργίες που απαιτούνται.
- Η επιλογή των κατάλληλων δομών δεδομένων είναι εξαιρετικά σημαντική για τη δημιουργία αποδοτικών προγραμμάτων.
- Πολλές κοινές δομές δεδομένων έχουν υλοποιηθεί στην STL (Standard Template Library) της C++.

Δομές Δεδομένων – ένα παράδειγμα

- Έστω ένα πρόγραμμα στο οποίο ένα μεγάλο σύνολο λέξεων αποθηκεύεται με κάποιο τρόπο.
  - Δεδομένης μιας λέξης, θέλουμε να γνωρίζουμε το που έχει αποθηκευτεί.
- Αν οι λέξεις είναι αποθηκευμένες σε έναν πίνακα ή σε μια συνδεδεμένη λίστα ή σε ένα διάνυσμα τότε θα πρέπει να περάσουμε από όλα τα δεδομένα (στη χειρότερη περίπτωση) έτσι ώστε να βρεθεί η θέση του στοιχείου που ψάχνουμε, δηλαδή η λειτουργία αυτή θα έχει πολυπλοκότητα χρόνου O(N).
- Μια καλύτερη δομή δεδομένων για το συγκεκριμένο πρόβλημα είναι ένας πίνακας κατακερματισμού που εντοπίζει το στοιχείο προς αναζήτηση σε χρόνο O(1).
- Σε μεγάλα σύνολα δεδομένων η χρήση πίνακα κατακερματισμού στη θέση δομών δεδομένων όπως οι πίνακες, οι συνδεδεμένες λίστες και τα διανύσματα επιτυγχάνουν επιτάχυνση του χρόνου εκτέλεσης της τάξης του 100.

Δομές Δεδομένων – ένα ακόμα παράδειγμα

- Έστω το πρόβλημα του χρονοπρογραμματισμού εργασιών που εκτελούνται σε μια CPU.
  - Καθώς συμβαίνει πολύ συχνά (~50 φορές ανά δευτερόλεπτο), πρέπει να υλοποιηθεί με τον πλέον αποδοτικό τρόπο.
- Ένας πίνακας αποθηκεύει τις πληροφορίες της κάθε εργασίας που πρέπει να εκτελεστεί μαζί με την προτεραιότητα της.
  - Ζητείται η επιλογή της εργασίας με την υψηλότερη προτεραιότητα κάθε φορά προκειμένου να εκτελεστεί.
  - Οι λειτουργίες που θα πρέπει να υποστηρίζονται είναι:
    - Εισαγωγή μιας νέας διεργασίας με δεδομένη προτεραιότητα στη δομή.
    - Αφαίρεση της διεργασίας με την υψηλότερη προτεραιότητα από τη δομή.
  - Πως μπορούν και οι δύο λειτουργίες να γίνουν αποδοτικά;
    - Ο πίνακας (είτε είναι ταξινομημένος είτε όχι) δεν αποτελεί αποδοτική λύση.
    - Η χρήση της δομής **Ουρά Προτεραιότητας** (priority queue) είναι ιδανική.
      - Η ουρά προτεραιότητας (σωρός μεγίστων) διατηρεί το στοιχείο με τη μεγαλύτερη προτεραιότητα σε θέση που μπορεί να προσπελάσει σε χρόνο O(1), ενώ η αφαίρεση του μέγιστου στοιχείου από την ουρά προτεραιότητας όπως και η προσθήκη ενός νέου στοιχείου ολοκληρώνεται σε χρόνο O(logN).

Αφηρημένοι τύποι δεδομένων (ADT=Abstract Data Types)

- Οι δομές δεδομένων συνήθως υλοποιούνται ως αφηρημένοι τύποι δεδομένων: στοίβες, ουρές, διανύσματα, συνδεδεμένες λίστες, δένδρα

- Στοίβες (stacks)
  - LIFO (Last In First Out). Οι εισαγωγές και οι διαγραφές επιτρέπονται μόνο από την κορυφή της στοίβας.
  - Μια στοίβα έχει δύο βασικές λειτουργίες:
    - push: προσθέτει ένα στοιχείο στην κορυφή της στοίβας
    - pop: αφαιρεί ένα στοιχείο από την κορυφή της στοίβας
  - Τυπικές περιοχές εφαρμογών των στοιβών είναι οι μεταγλωττιστές, τα λειτουργικά συστήματα, η διαχείριση της μνήμης του προγράμματος κατά την κλήση συναρτήσεων κ.α.

Ουρές (queue)

- FIFO (First In First Out): Οι εισαγωγές γίνονται στο πίσω άκρο της ουράς, και οι διαγραφές γίνονται από το εμπρός άκρο της ουράς.
- Μια ουρά έχει δύο βασικές λειτουργίες:
  - enqueue: προσθήκη ενός στοιχείου στην ουρά
  - dequeue: αφαίρεση ενός στοιχείου από την ουρά
- Τυπικές περιοχές εφαρμογών στις οποίες χρησιμοποιούνται ουρές είναι η ουρά αναμονής του εκτυπωτή, ο προγραμματισμός εργασιών στα λειτουργικά συστήματα κ.α.

Διανύσματα (vectors)

- Τα διανύσματα είναι δομές δεδομένων που αποθηκεύουν δεδομένα του ίδιου τύπου και βασίζονται στην αποθήκευσή τους σε κάποιο πίνακα.
- Ενθυλακώνοντας έναν πίνακα σε μια κλάση, μπορούμε:
  - να χρησιμοποιήσουμε δυναμική δέσμευση μνήμης έτσι ώστε να αυξομειώνουμε το μέγεθος του πίνακα εφόσον χρειάζεται.
  - να χειριζόμαστε απόπειρες πρόσβασης εκτός των ορίων του πίνακα.
- Πλεονέκτημα: τυχαία πρόσβαση (πρόσβαση απευθείας σε κάποιο στοιχείο με βάση το δείκτη).
- Μειονεκτήματα: Οι εισαγωγές και οι διαγραφές ενδιάμεσα στη δομή είναι αργές καθώς μπορεί να απαιτούν την ολίσθηση πολλών στοιχείων που καταλαμβάνουν διαδοχικές θέσεις στον πίνακα.

Συνδεδεμένες λίστες (linked lists)

- Οι συνδεδεμένες λίστες αποτελούν συλλογές δεδομένων που συνδέονται με τη χρήση δεικτών, σε μια σειρά.
- Αποτελούνται από συλλογές «κόμβων», που δημιουργούνται από μια αναδρομική κλάση (ή struct).
  - Αναδρομική κλάση: μια κλάση που τα μέλη δεδομένων της περιέχουν τουλάχιστον ένα δείκτη που δείχνει προς ένα αντικείμενο της ίδια κλάσης.
  - Κάθε κόμβος περιέχει κάποια δεδομένα, και έναν δείκτη προς το επόμενο κόμβο (στην απλά συνδεδεμένη λίστα)
  - Οι κόμβοι μπορούν να βρίσκονται οπουδήποτε στη μνήμη και όχι υποχρεωτικά σε συνεχόμενες θέσεις μνήμης όπως στους πίνακες.
  - Οι κόμβοι συνήθως δεσμεύονται δυναμικά, συνεπώς μια συνδεδεμένη λίστα μπορεί να γίνει όσο μεγάλη απαιτείται.
- Πλεονεκτήματα: Οι εισαγωγές και οι διαγραφές είναι συνήθως γρήγορες. Απαιτούν τη δημιουργία ενός νέου κόμβου και την αλλαγή κάποιων δεικτών.
- Μειονεκτήματα: Δεν υποστηρίζουν τυχαία πρόσβαση. Ο εντοπισμός ενός στοιχείου της λίστας απαιτεί τη διάσχιση της λίστας.
- Παρατηρήστε ότι τα πλεονεκτήματα των διανυσμάτων αποτελούν μειονεκτήματα των συνδεδεμένων λιστών και αντίστροφα.

Δένδρα (trees)

- Τα δένδρα είναι μη-γραμμικές συλλογές δεδομένων, που συνδέονται με δείκτες (όπως οι συνδεδεμένες λίστες).
- Αποτελούνται από κόμβους (με αναδρομικούς ορισμούς). Κάθε κόμβος μπορεί να περιέχει δύο ή περισσότερους δείκτες προς άλλους κόμβους.
- Τυπική περίπτωση δένδρου είναι τα δυαδικά δένδρα (Binary Trees):
  - Κάθε κόμβος περιέχει ένα στοιχείο δεδομένων, και δύο δείκτες που καθένας δείχνει προς έναν άλλο κόμβο.
  - Είναι ιδιαίτερα χρήσιμα για τη γρήγορη αναζήτηση καθώς και για τη διατήρηση δεδομένων σε ταξινομημένη σειρά.
  - Η αναζήτηση σε ένα δυαδικό δένδρο αναζήτησης (BST=Binary Search Tree) συνίσταται στην εύρεση μιας διαδρομής στο δένδρο, που ξεκινά από τη ρίζα του δένδρου και σε κάθε επιλογή μονοπατιού (επιλογή αριστερού ή δεξιού κόμβου) απαλείφει τις μισές από τις εναπομείνασες αποθηκευμένες τιμές της δομής.

Ερωτήσεις σύνοψης

- Ποιος είναι ο ρόλος των `argc` και `argv` μεταβλητών στην `main`;
- Τι εξυπηρετεί η δήλωση `#pragma once` σε ένα αρχείο header;
- Πως αλλιώς μπορεί να γραφεί ισοδύναμος κώδικας με τη δήλωση `#pragma once` ;
- Ποιο είναι το αποτέλεσμα της εντολής `56 << 1`;
- Πότε η συνδεδεμένη λίστα αποτελεί καλύτερη λύση σε σχέση με το διάνυσμα;
- Μπορεί να κατασκευαστεί τελεστής τυχαίας προσπέλασης `[]` για μια συνδεδεμένη λίστα;

Αναφορές

- http://www.cs.fsu.edu/~xyuan/cop3330/