Programare Orientată Obiect

Laborator 1

Programarea orientată obiect

- Numită și programare orientată pe obiecte, a apărut din necesitatea exprimării problemei într-un mod mai natural ființei umane (în engleză OOP – Object Oriented Programming)
- Astfel unitățile care alcătuiesc un program se apropie mai mult de modul nostru de a gândi decât de modul de lucru al calculatorului
- Până la apariția programării orientate pe obiect programele erau implementate în limbaje de programare procedurale (C, Pascal)
- Considerată unul din cei mai importanți pași făcuți în evoluția limbajelor de programare, permite separarea unităților de cod în entități abstracte (numite clase)

Ce trebuie să știi deja

- Programare structurală (procedurală) în C (conceptul de C++ a fost introdus tocmai după ce limbajului de programare C i-a fost adăugat suportul pentru POO)
- Mediul de lucru Visual Studio (recomand versiunea 2012, dar și cele mai vechi sunt ok)
- Dacă ești student eligibil Microsoft DreamSpark poți downloada VS gratuit de pe platformă
- Dacă nu, poți downloada versiunea Visual C++ 2010 Express

Avantajele POO

- Codul devine mai lizbil, mai ușor de înțeles și depanat
- Putem crea programe mari foarte ușor, reușind să gândim codul corect încă de la început
- Prin abstractizare putem grupa codul corespunzător unui concept și totodată îl putem separa de restul codului
- Prin încapsulare (ascunderea unor date pentru anumite entități) putem evita accesul necontrolat la date
- Vom mai descoperi pe parcurs…

Despre ce vorbim azi

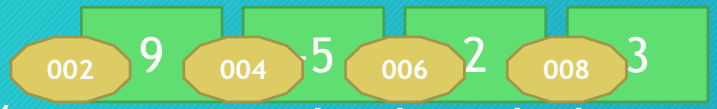
- Conceptul de pointer (recapitulare)
- Operatori specifici pointerilor
- Transmiterea parametrilor
- Exerciții

Conceptul de pointer

- Variabilă ce reține adresa unei alte variabile
- Putem accesa o dată necesară dacă îi cunoaștem adresa
- Exemplu: Un poștaș aruncă în fiecare dimineață ziare în diverse curți ce au abonament la o anumită publicație. Poștașul nu poate ști direct dacă un abonat a citit sau nu publicația din acea zi. Dar știind adresa la care a lăsat-o poate merge și întreba abonatul dacă a făcut acest lucru sau nu
- Similar se manifestă și pointerii

Conceptul de pointer

- Să spunem că aceasta este memoria



- După cum știm limbajul de programare C împarte memoria ce poate fi folosită în două zone ce au comportament diferit: **stack** (sau stiva internă unde se salvează variabilele de tip fundamental și unde alocarea memoriei sa face automat) și **heap** (sau stiva program unde salvăm de obicei pointerii și alocarea se face manual)
- În imaginea de mai sus avem o porțiune din stack unde s-au salvat 4 variabile întregi ce au valorile 9, -5, 2, respectiv 3 la adresele 002, 004, 006 și 008 (adresele sunt fictive, cele reale din RAM sunt în hexa și au forma 0x456783)

Conceptul de pointer

- Dacă vrem să salvăm adresa la care este variabila 2, nu valoare ei

- Atunci declarăm un pointer la întreg (pentru că acesta este tipul valorii reținute la acea adresă pe care îl inițializăm cu valorea corespunzătoare)
- Avem așadar:

```
int x = 2; //instrucțiune ce a fost scrisă anterior astfel încât s-a produs salvarea lui 2 în stack
int *px; //declarăm un pointer la întreg
px = &x; //îi atribuim adresa la care este salvată valoarea 2
```

- În acest fel în px vom avea de fapt 006, iar după dereferențiere (aflarea conținutului, vom avea 2)

Operatori specifici pointerilor

- Operatorul * ce are dublu rol: definirea unui pointer și extragerea conținutului de la o adresă
- Ex: int *px; //definirea pointerului px la întreg
- Ex: cout<<(*px); //afișarea valorii salvată la adresa pointată
- Operatorul & ce are rolul de a extrage adresa la care este salvată o variabilă
- Operatorii +, -, ++, -
- Pointerii sunt de tip **near** (situați în același segment de memorie, ocupă 2 octeți) sau **far** (în segmente diferite, ocupă 4 octeți)

Definire/dezalocare

- Folosind **malloc** și **free** (nerecomandate în POO)
- Folosind **new** și **delete**
- Exemplu definirea unui pointer la float, alocarea spațiului de memorie corespunzator și dezalocarea lui:

```
float *px =  new float();
delete px;
```

Pointerul NULL semnifică faptul că adresa referită nu este validă.

Transmiterea parametrilor

- Prin valoare: se creează copii ale parametrilor de apel, valorile originale rămân neschimbate. Ex: void add(int x, int y)
- Prin referință: parametri de apel vor fi schimbați de funcție. Ex: void add(int &x, int y)
- Prin adresă: asemănător cu transferul prin referință doar că va trebui să ne ocupăm personal de referențiere/dereferențiere. Ex: void add(int *x, int y)

Exerciții

- Să trecem la treabă! ☺