Introducere si concepte de baza

1. Necesitatea acestei discipline

Evolutia tehnicilor de programare determina realizarea unor produse program caracterizate prin complexitate ridicata si prin consumuri de resurse reduse.

Activitatea programului devine, odata cu eliminarile restrictiilor impuse de sistemele de calcul, o activitate de alocare si nivelare a resurselor software.

Dintr-o multitudine de limbaje, medii de programare si biblioteci de programe trebuie alese si asamblate acele componente care conduc la produse program performante. Pentru efectuarea unei alegeri corespunzatoare, resursele trebuie cunoscute în cele mai mici detalii.

Într-un context mai larg, ***structurile de date se constituie ca resurse la dispozitia programatorilor***, care prin diversitate ***influenteaza hotarâtor calitatea programelor***. Diferenta dintre o aplicatie, care este conceputa pentru a nu utiliza fisiere, si aceeasi aplicatie, care utilizeaza fisiere, se reflecta la nivelul costurilor prin câstig sau pierdere, dupa cum solutia aleasa este sau nu adecvata.

Rezolvarea unei probleme începe cu definirea structurilor de date, continua cu ***utilizarea acestora*** si se încheie cu ***stocarea rezultatelor prelucrarii***, tot sub forma unor structuri de date.

Studierea structurilor de date revine la clasificarea datelor, a operatiilor posibile cu fiecare tip de date, în asa fel încât realizarea si dezvoltarea programelor sa devina avantajoasa atât pentru programator cât si pentru utilizator.

Exista doua modalitati distincte de a analiza structurile de date:

- abordarea logica, filozofia de realizare, formalizare si de transformare;
- construirea efectiva a structurilor, utilizând resursa memorie calculator împreuna cu algoritmii de încarcare si de adresare.

Fiecarui tip de date si mod de structura îi corespund anumite situatii în care utilizarea conduce la timp de acces la informatie si la necesar de memorie, reduse. Astfel, apar operatii de prelucrare suplimentare, precum conversii, citiri si scrieri care micsoreaza viteza de obtinere a rezultatelor, conducând la cresterea costului prelucrarii.

Limbajele de programare, în marea lor diversitate, se aseamana prin tipurile de date pe care programatorii le utilizeaza. Trecerea de la un limbaj la altul în conditiile cunoasterii caracteristicilor generale ale structurilor de date devine posibila, iar efortul cerut este minim. Într-o acceptiune mai restrânsa ***structurile de date iau în considerare operanzii***. Astfel, operanzii sunt constante, variabile simple, masive, structuri de tip articol, fisierele precum si structurile de date care se construiesc dinamic ca de exemplu, listele si arborii.

Dezvoltarile teoretice au menirea de a oferi modele pentru fiecare structura de date si pentru a permite considerarea unora mai simple dintre ele drept cazuri particulare ale altor structuri. Generalizarile conduc la includerea chiar a programului în categoria structurilor de date.

Structurile de date sunt strict legate de programarea intr-un limbaj de programare. Daca se considera structurile de date S_1, S_2, S_3, ..., S_n, pentru o problema PROB se scriu n variante de program, fiecare avand cate o structura de date dominanta. Se vor scrie, respectiv, programele $PROG_1$, $PROG_2$, ..., $PROG_n$.

In C# sunt definite clase de tip colectie pentru a facilita lucrul cu structuri de date dinamice, precum: lista, stiva, coada, etc.

In MFC exista clase specializate, ca de exemplu:

CArray – pentru lucrul cu masive;

CList – pentru lucrul cu liste, stive, cozi;

CMap – pentru implementarea structurii de hash (tabela de dispersie).

In toate implementarile s-a urmarit asigurarea unei ***cat mai mari generalitati*** si a unei ***cat mai mari flexibilitati. In C++, generalitatea este asigurata prin faptul ca se pot defini sabloane de clase (clase template), iar flexibilitatea se asigura prin scrierea unor secvente de cod, trimise ca parametri altor functii, prin intermediul pointerilor la functii***. C#, fiind un limbaj pur obiectual, orice obiect poate fi manipulat si ca tip "object". ***Prin manipularea variabilelor de tip object, in limbajul C# se asigura o maxima generalitate colectiilor de date.***

In C#, urmatoarele clase de tip colectie implementeaza principalele structuri de date dinamice:

ArrayList – implementeaza structura de lista;

Stack – implementeaza structura de stiva;

Queue – implementeaza structura de coada;

HashTable – implementeaza structura de hash.

Necesitatea studierii disciplinei de structuri de date se impune prin:

- alegerea celei mai adecvate structuri de date;
- efortul de programare sa fie cat mai mic;
- programul sa conduca la durate ale tranzactiilor cat mai reduse;
- programul sa fie cat mai usor de intretinut;
- depanarea sa necesite eforturi mici;
- cunoasterea proprietatilor fiecarei structuri de date;
- construirea si utilizarea de biblioteci de proceduri pentru fiecare structura.

Daca se cunosc toate tipurile de structuri de date, proprietatile si, mai ales, efectele utilizarii lor asupra eficientei programului, atunci in mod evident:
- se construiesc un numar foarte restrans de variante;
- se alege varianta cea mai performanta dintr-un numar foarte restrans de variante;
- se fac cheltuieli mult mai mici.

Cunoasterea structurilor de date permite programatorilor sa priveasca procesul de dezvoltare software ca proces de alocare si nivelare de resurse. Instructiunile sunt resurse, limbajele de programare sunt resurse, fisierele si organizarile lor sunt resurse, bibliotecile de subprograme, bibliotecile de clase, sunt tot resurse.

Este important ca programatorul sa aleaga acele resurse care conduc spre programe eficiente la:
- nivelul firmei care dezvolta software;
- utilizator prin maximizarea gradului de satisfactie a acestuia.

2. Obiectivul disciplinei

Despre structuri de date se invata la multe discipline, precum:
- limbaje de programare;
- birotica;
- sisteme de calcul;
- baze de date;
- programe aplicative.

Sistematizarea cunostintelor insa este efectuata numai la ***disciplina STRUCTURI DE DATE care are ca obiectiv prezentarea tuturor structurilor de date cu proprietatile, avantajele si dezavantajele utilizarii lor, in vederea utilizarii lor eficiente.***

Pentru aceasta trebuie sa se:
- enumere structurile;
- clasifice structurile de date;

- prezinte fiecare structura;
- construiasca proceduri pentru implementarea operatiilor;
- dezvolte metrici ale structurilor de date;
- arate cum se agrega structurile de date;
- clarifice ce inseamna utilizare eficienta;
- arate care sunt limitele procesului de optimizare.

3. Criterii de clasificare a structurilor de date

Dupa ***criteriului alocarii memoriei*** exista structuri de date:
- ***statice (masive, articol, fisier);***
- ***dinamice (liste, stive, cozi, arbori).***

Dupa ***numarul pointerilor de referire elemente*** exista structuri cu:
- ***zero pointeri;***
- ***un pointer;***
- ***doi pointeri;***
- ***mai multi pointeri.***

Dupa ***disciplina de parcurgere***:
- ***LIFO (Last In First Out);***
- ***FIFO (First In First Out);***
- ***RSD (Radacina-Stanga-Dreapta);***
- ***SDR (Stanga-Dreapta-Radacina);***
- ***DSR (Dreapta-Stanga-Radacina).***

4. Modele de prezentare a structurilor de date

- ***modelul grafic*** presupune: definire zone de memorie si legaturi cu arce orientate;
- ***modelul analitic*** presupune utilizarea funtiilor *tip(), prec(), succ(), cont(), adr(), in(), out()*;
- ***modelul graf*** utilizeaza noduri si arce;
- ***modelul textual*** utilizeaza cuvinte din limbajul natural;
- ***modelul sintactic*** utilizeaza conventiile limbajelor de programare.

5. Operatii cu structuri de date

Indiferent care sunt structurile de date, se definesc urmatoarele operatii:
- creare element;
- adaugare element;
- traversare structura;
- concatenare doua sau mai multe structuri de acelasi tip;
- cautare element dupa cheie sau pozitie;
- interschimb elemente adiacente;
- interschimb elemente oarecare;
- manipulare de legauri;
- manipulare de informatii utile;
- stergere element;
- stergere structura;
- numarare elemente;
- modificare campuri;
- inserare element dupa pozitie;
- inserare element cu pastrarea unei proprietati;

- copierea de informatii utile;
- echilibrare;
- sortare fara mentinerea vechilor legaturi;
- sortare cu mentinerea vechilor legaturi;
- compunere de structuri de acelasi tip.

Pentru fiecare structura se construieste o biblioteca de proceduri care implementeaza aceste operatii, in forma nerecursiva sau in forma recursiva.