**PHP ACADEMY**

**CURS PHP**

**METODE DE PROIECTAREA ALGORITIMILOR PROIECTAREA STRUCTURATĂ**

**STUDENT: Moldovai Gianina**

**ORADEA**

**2024**

1. **METODE DE PROIECTAREA ALGORITIMILOR – PROIECTAREA STRUCTURATĂ**

Proiectarea algoritmilor este un proces esențial în dezvoltarea de software și implică conceperea și elaborarea unor secvențe precise de instrucțiuni care să rezolve o anumită problemă sau să îndeplinească un anumit obiectiv. Una dintre metodele fundamentale de proiectare a algoritmilor este proiectarea structurată, care se bazează pe principii clare și metode bine definite pentru a crea algoritmi eficienți și ușor de înțeles. Iată câteva principii esențiale ale proiectării structurate a algoritmilor:

1. Dezvoltarea modulară:

Principiul dezvoltării modulare implică împărțirea problemei mari în subprobleme mai mici și mai ușor de gestionat. Aceste subprobleme sunt rezolvate separat și apoi integrate pentru a obține soluția finală.

1. 2. Utilizarea structurilor de control:

Proiectarea structurată implică utilizarea judicioasă a structurilor de control, cum ar fi buclele (for, while) și instrucțiunile condiționale (if, else), pentru a controla fluxul de execuție al algoritmului.

1. 3. Definirea clară a datelor și a operațiilor:

Este important să se definească clar tipurile de date și operațiile disponibile pentru acestea înainte de a proiecta algoritmul. Aceasta include stabilirea tipurilor de date pentru datele de intrare și de ieșire și definirea funcțiilor și a operațiilor care vor fi aplicate asupra acestora.

1. 4. Modularitatea și coeziunea:

Principiul modularității implică împărțirea algoritmului în module sau funcții logice, fiecare având o responsabilitate bine definită. Aceste module ar trebui să fie coezive, adică să conțină instrucțiuni și date relevante pentru sarcina lor specifică.

1. 5. Structuri de date eficiente:

Alegerea și utilizarea structurilor de date corespunzătoare sunt esențiale pentru proiectarea unui algoritm eficient. Este important să se selecteze structurile de date care să faciliteze accesul și manipularea datelor în mod eficient.

1. 6. Documentarea și comentariile:

Documentarea și adăugarea de comentarii în codul algoritmului sunt esențiale pentru a face algoritmul ușor de înțeles și de întreținut. Acestea ar trebui să ofere informații clare despre funcționarea algoritmului și despre rolul fiecărei părți a acestuia.

1. 7. Testarea și verificarea:

În timpul proiectării algoritmilor, este important să se efectueze teste și verificări pentru a asigura că algoritmul funcționează așa cum este de așteptat și că rezolvă problema propusă în mod corect și eficient.

Proiectarea structurată a algoritmilor încurajează un proces de gândire clar și organizat și ajută la crearea de algoritmi corespunzători, ușor de înțeles și de întreținut. Prin aplicarea acestor principii, dezvoltatorii pot crea soluții eficiente și fiabile pentru o varietate de probleme și aplicații.

**INSTRUCȚIUNILE SIMPLE**

Instrucțiunile simple sunt cele mai de bază construcții ale unui limbaj de programare și sunt folosite pentru a exprima operații individuale sau acțiuni în cadrul unui program. Acestea sunt fundamentele din care sunt construite structurile mai complexe, cum ar fi buclele, instrucțiunile condiționale și subprogramele. Iată câteva exemple de instrucțiuni simple care sunt comune în majoritatea limbajelor de programare:

* **Instrucțiunea de atribuire** este folosită pentru a asigna o valoare unei variabile. Exemplu: x=10;
* **Instrucțiunile pentru intrare/ieșire** sunt folosite pentru a comunica cu utilizatorul sau cu alte dispozitive.

Exemplu intrare: scanf("%d", &x);

Exemplu ieșire: printf("Valoarea lui x este: %d\n", x);

* **Instrucțiunile de control al fluxului** sunt folosite pentru a controla fluxul de execuție al programului.

Exemplu: if (x > 10) {

// Blocul de cod care se execută dacă condiția este adevărată

} else {

// Blocul de cod care se execută dacă condiția este falsă

}

* **Instrucțiunile pentru manipularea datelor** sunt folosite pentru a efectua diverse operații pe date.

Exemplu: x++;

**STRUCTURILE DE DECIZIE**

O structură de decizie este o construcție care permite programului să ia decizii în funcție de o anumită condiție sau condiții. Aceasta permite programului să execute diferite blocuri de cod în funcție de valoarea unei expresii sau a unei variabile. Există mai multe tipuri de structuri de decizie în programare, cum ar fi declarațiile if, if-else, switch și operatorul ternar. Aceste structuri de decizie sunt utilizate pentru a controla fluxul programului și pentru a face alegeri bazate pe condiții specifice.

În programare, există trei tipuri de structuri de decizie/alternativă comune:

1. Structura de decizie cu o singură ramură:

Această structură de decizie este reprezentată de declarația "if". Programul verifică o singură condiție și execută un bloc de cod dacă condiția este adevărată. Dacă condiția este falsă, programul trece peste blocul de cod și continuă execuția.

Exemplu în limbajul PHP:

if ($var > 3) { echo "Variabila este mai mare decât 3."; }

1. Structura de decizie cu două ramuri:

Această structură de decizie este reprezentată de declarația "if-else". Programul verifică o condiție, iar dacă aceasta este adevărată, se execută un bloc de cod, iar dacă este falsă, se execută un alt bloc de cod.

Exemplu în limbajul PHP:

if ($var > 3) { echo "Variabila este mai mare decât 3."; }

else { echo "Variabila este mai mică sau egală cu 3."; }

1. Structura de decizie cu selecție multiplă (case/switch):

Această structură de decizie permite programului să evalueze o expresie și să execute diferite blocuri de cod în funcție de valoarea expresiei. Fiecare valoare posibilă a expresiei este asociată cu un bloc de cod specific.

Exemplu în limbajul PHP:

switch ($var) {

case "rosu":

echo "Culoarea este roșie.";

break;

case "verde":

echo "Culoarea este verde.";

break;

case "albastru":

echo "Culoarea este albastră.";

break;

default:

echo "Culoarea nu este niciuna dintre opțiunile disponibile."; }

**STRUCTURI REPETITIVE**

În programare, există trei tipuri de structuri repetitive comune:

1. Structura repetitivă cu test initial:

Această structură repetitivă este reprezentată de declarația "while". Programul verifică o condiție la începutul fiecărei iterații și execută blocul de cod atâta timp cât condiția este adevărată.

Exemplu în limbajul PHP:

$i = 0;

while ($i < 10) {

echo $i;

$i++;

}

2. Structura repetitivă cu test final**:**

Această structură repetitivă este reprezentată de declarația "do-while". Programul execută întâi blocul de cod și apoi verifică condiția la sfârșitul fiecărei iterații. Dacă condiția este adevărată, programul continuă să execute blocul de cod.

Exemplu în limbajul PHP:

$i = 0;

do {

echo $i;

$i++;

} while ($i < 10);

1. Structura repetitivă cu număr cunoscut de pași:

Această structură repetitivă este reprezentată de declarația "for". Programul execută un număr cunoscut de iterații și verifică condiția la începutul fiecărei iterații.

Exemplu în limbajul PHP:

for ($i = 0; $i < 10; $i++) {

echo $i;

1. Scrierea de programe echivalente:

În multe cazuri, există mai multe moduri de a scrie un program echivalent, adică un program care face același lucru, dar folosind o altă sintaxă. De exemplu, cele trei exemple de mai sus pot fi scrise astfel:

Exemplu în limbajul PHP:

// Structura repetitivă cu test initial

$i = 0;

while ($i < 10):

echo $i;

$i++;

endwhile;

// Structura repetitivă cu test final

$i = 0;

do {

echo $i;

$i++;

} while ($i < 10);

// Structura repetitivă cu număr cunoscut de pași

for ($i = 0; $i < 10; $i++):

echo $i;

endfor;

Aceste structuri repetitive permit programatorilor să execute blocuri de cod de mai multe ori și să automatizeze sarcini repetitive în limbajul PHP.