**11. Imperatív Nyelvek**

Az imperatív nyelvek olyan programozási nyelveket jelentenek, amelyek olyan utasításokból állnak, amelyek meghatározzák, hogy a számítógépnek pontosan mit kell tennie. Ezek az utasítások általában adatok manipulálására, változók állapotának módosítására vagy vezérlési szerkezetek (pl. ciklusok, elágazások) végrehajtására vonatkoznak. Az imperatív nyelvek közé tartoznak például a C, C++, Java, Python, és sok más programozási nyelv, melyek egyaránt alacsony és magas szintűek is lehetnek.

**Típusok**

Az első pont az imperatív nyelvek tételben a típusokról szól. Az imperatív nyelvekben a típusok alapvetően az adatok jellegét és azon műveletek típusát, amelyeket ezekkel az adatokkal végezhetünk határozzák meg. Az adattípusok meghatározása fontos az adatok helyes kezelése és manipulálása szempontjából. Meghatározzák, hogy milyen műveleteket lehet végrehajtani az adott típusokon, és segítenek a programokban a hibák kiküszöbölésében, mivel a típushiba elkerülhetővé teszi a nem megfelelő típusú adatok keveredését. Az adattípusok a programozó számára is segítséget nyújtanak az adatok strukturálásában és azok megfelelő kezelésében. Az imperatív nyelveken belül az alábbi típusokat különböztetjük meg:

* **Alapvető adattípusok:** Ezek az adattípusok az alapvető adatokat képviselik, mint például egész számok, lebegőpontos számok, karakterek és logikai értékek. Az említett típusok általában a nyelv szintaxisában beépítve vannak, és a nyelv szabványa részét képezik.
  + **Integer:**

Az egész számok tárolására szolgál. Méretük általában 32 vagy 64-bites.

* + **Float:**

A lebegőgombos számok tizedes törtszámokat tárolnak, egyszeres vagy dupla pontossággal. Egyszeres pontossággal 32 bitet foglalnak és körülbelül 7 számjegy reprezentálja a tizedes törtszámokat. Dupla pontossággal pedig 64 bitet foglalnak, továbbá 15-16 számjegy jelenik meg.

* + **Char:**

Egyetlen karaktert tárolnak. Ezek a karakterek lehetnek betűk, számok, írásjelek vagy más speciális karakterek, mint például szóköz vagy új sor.

* + **String:**

A sztringek karakterláncokat vagy szöveges adatokat tárolnak.

* + **Boolean:**

A logikai értékek igaz vagy hamis értékeket reprezentálnak, amelyek a logikai műveletekhez és kifejezésekhez használatosak, továbbá általában egyetlen biten tárolódnak (0, ha hamis / 1, ha igaz).

* **Összetett adattípusok:** Ezek az adattípusok lehetővé teszik az adatok összetett struktúrában történő tárolását, például tömbök, listák, rekordok és struktúrák. Ezek az adattípusok segítenek a programozónak a strukturált adatok kezelésében, amelyek több összetevőből állnak.
  + **Arrays (tömbök):**

A tömbök olyan adatszerkezetek, amelyek egy sor vagy egydimenziós tartományt képeznek, amelyben azonos típusú elemek vannak tárolva. A tömbök lehetővé teszik a programok számára, hogy hatékonyan tároljanak és manipuláljanak azonos típusú adatokat, például egész számokat vagy karaktereket.

* + **Lists:**

A listák lehetővé teszik az adatok dinamikus tárolását és kezelését. Ezek az adatszerkezetek dinamikusan változó méretűek lehetnek, és különböző műveleteket támogatnak, mint például elemek hozzáadása, eltávolítása vagy keresése.

* + **Struct:**

A struktúrák lehetővé teszik a programok számára, hogy egyszerre tároljanak különböző típusú adatokat egyetlen összetett adatszerkezetben. A struktúrákban különböző mezők vagy tagok vannak, amelyek különböző típusú adatokat tárolnak.

* + **Records:**

A rekordok hasonlóak a struktúrákhoz, de általában az adattagok eléréshez használt műveletekkel együtt vannak lehetővé téve a programok számára, hogy könnyen hozzáférjenek az adattagokhoz és manipulálják azokat.

* **Felhasználó által definiált típusok:** Sok imperatív nyelv lehetővé teszi a felhasználók számára, hogy saját típusokat definiáljanak. Ez általában az osztályok létrehozásával és az objektumorientált programozás során lehetséges, ahol a fejlesztők saját típusokat hozhatnak létre az alkalmazásukban szereplő specifikus adatok reprezentálására.

**Aritmetikai, relációs és logikai műveletek**

Az aritmetikai, relációs és logikai műveletek a programozási nyelvek alapvető műveleteire utal, amelyek lehetővé teszik az adatok közötti számításokat, összehasonlításokat és logikai műveleteket. Ezen műveletek kombinálásával számos összetett számítást és döntést lehet létrehozni a programok készítése során. Az aritmetikai műveletek segítségével matematikai számításokat végezhetünk el, a relációs műveletekkel adatokat hasonlíthatunk össze, a logikai műveletekkel pedig logikai kifejezéseket létrehozásához van lehetőségünk. Ezek az alapvető műveletek alkotják az imperatív nyelvek nyelvi alapját, és széleskörű alkalmazásuk révén segítik a programok hatékony és pontos működését.

* **Aritmetikai műveletek:** Az aritmetikai műveletek az alapvető matematikai műveleteket jelentik, mint például az összeadást, kivonást, szorzást és osztást. Lehetővé teszik az adatok matematikai manipulációját, amelyek elengedhetetlenek a számítások végrehajtásában. Az összeadás és kivonás, továbbá a szorzás művelet jelölése a matematikában használttal megegyezik. Az osztást kétféleképpen jelöljük attól függően, hogy a kerekített eredményt vagy az osztás maradékát szeretnénk megkapni:
  + **Osztás:**

jelölés: /

pl.: int a = 10;

int b = 3;

int result = a / b;

Az eredmény: 3 (lefelé kerekítés).

* + **Maradékos osztás:**

jelölés: %

pl.: int a = 10;

int b = 3;

int result = a % b;

Az eredmény: 1 (10 osztva 3-mal az 3 és maradék az 1).

* **Relációs műveletek:** A relációs műveletek az imperatív programozási nyelvekben olyan operátorok, amelyek segítségével összehasonlításokat végezhetünk két vagy több érték között, és különböző állításokat vagy feltételeket állíthatunk fel az alapján, hogy az összehasonlítás igaz vagy hamis. Ezek az operátorok fontosak az elágazásokban és ciklusokban, valamint az adatok rendezésében és keresésében, ahol igaz összehasonlítás esetén hajtódnak végre a benne levő műveletek. Néhány fontos relációs művelet az imperatív nyelvekben:
  + **Egyenlőség (==)**
  + **Nem egyenlőség (!=)**
  + **Nagyobb (>), kisebb (<)**
  + **Nagyobb vagy egyenlő (>=), kisebb vagy egyenlő (<=)**
* **Logikai műveletek:** Ezek a műveletek a logikai értékeken (igaz vagy hamis) végeznek műveleteket. Példák erre az ÉS (AND), VAGY (OR), és NEM (NOT) műveletek. A logikai műveletek segítenek a feltételek kifejezésében és a programok logikai viselkedésének irányításában.
  + **És (&&):**

Az „És” operátor két logikai érték közötti konjunkciót (logikai „És”) jelöl, és igazat ad vissza, ha mindkét operandus igaz, ellenkező esetben hamisat. pl.:

bool a = true;

bool b = false;

if (a && b) {

// Ez a kód nem fut le, mert mindkét feltételnek igaznak kell lennie

}

* + **Vagy (||):**

A „Vagy” operátor két logikai érték közötti diszjunkciót (logikai „Vagy”) jelöl, és igazat ad vissza, ha legalább az egyik operandus igaz, ellenkező esetben hamisat. pl.:

bool a = true;

bool b = false;

if (a || b) {

// Ez a kód lefut, mert legalább az egyik feltétel igaz

}

* + **Nem (!):**

A „Nem” operátor egy logikai értéknek a negációját (logikai „Nem”) jelöli, és igaz értéket ad vissza, ha az operandus hamis, és hamisat ad vissza, ha az operandus igaz. pl.:

bool a = true;

if (!a) {

// Ez a kód nem fut le, mert az !a hamis, mert az a igaz

}

**Vezérlési szerkezetek**

A vezérlési szerkezetek a programok vezérléséért és az utasítások végrehajtásának irányításáért felelős alapvető szerkezeteket jelenti az imperatív nyelvekben. Ezek a vezérlési szerkezetek kulcsfontosságúak az algoritmikus gondolkodásban és a programozásban, mivel segítenek a programoknak logikus lépéseket követni, és hatékonyan irányítani az utasítások végrehajtását. Az egyes szerkezetek megfelelő használata lehetővé teszi a programok hatékonyabb és olvashatóbb kódjának írását, valamint segít a hibák elkerülésében és a programok megértésében.

* **Feltételes szerkezetek:** A feltételes szerkezetek az imperatív programozási nyelvekben olyan konstrukciók, amelyek lehetővé teszik a programok számára, hogy egy adott feltétel függvényében különböző utasításokat hajtsanak végre. Ezek a szerkezetek alapvető fontosságúak a programokban, mivel lehetővé teszik a döntéseket és a vezérlést az adott helyzeteknek megfelelően.
  + **If-else szerkezet:**

Az „If-else” szerkezet lehetővé teszi a programok számára, hogy különböző utasításokat hajtsanak végre egy adott feltétel igaz vagy hamis eredménye alapján. Ha a feltétel igaz, akkor az If ág fut le, más esetben pedig az Else ág. pl.:

int x = 10;

if (x > 0){

printf("x pozitív");

}

else{

printf("x nem pozitív");

}

* + **Else-if szerkezet:**

Az „Else-if” szerkezet lehetővé teszi a programok számára, hogy több feltételt is vizsgáljanak egymás után, és a megfelelő ágat válasszák ki. pl.:

int x = 0;

if (x > 0){

printf("x pozitív");

}

else if (x < 0){

printf("x negatív");

}

else{

printf("x nulla");

}

* + **Switch-case szerkezet:**

A „switch-case” szerkezet több választható ágat kínál a különböző feltételekhez. Az egyes feltételeket a case ágakba szervezik, és minden ágat külön-külön kezelnek. pl.:

int honap;

switch(honap){

case 1: printf("január\n"); break;

case 2: printf("február\n"); break;

…

case 12: printf("december\n"); break;

default: printf("Hibas hónap\n"); break;

}

* **Feltétel nélküli vezérlésátadás:** Ezek a szerkezetek lehetővé teszik a programnak, hogy egy adott ponttól függetlenül továbblépjen az utasítások végrehajtásában. pl.:
  + **Break:**

A break hatására megszakad a switch, illetve while, do vagy for utasítás végrehajtása, és a következő utasításra lép a vezérlés.

* + **JMP:**

Szintaxisa: JMP cimke

Hatására a vezérlés a megadott cimke paraméterre kerül. pl.:

MOV EAX, 2

JMP oda; Ugrás az "oda" cimkére

MOV EAX, 0; Ez az utasítás kimarad, mert átugrottuk

* + **Eljáráshívás:**

Szintaxisa: CALL eljárásnév

A CALL hatására az EIP megnövelt értéke a verembe kerül mentésre, majd a vezérlés átkerül az eljárás elejére.

* + **Visszatérés:**

A call utasítás párja.

Szintaxisa: RET

A RET utasítás a verem tetején levő értéket az EIP-be helyezi, ezzel visszaállítva az EIP korábbi értékét, és a vezérlést visszaadva a hívó helyére.

* **Ciklusok:** A ciklusok lehetővé teszik a programnak, hogy bizonyos utasításokat ismételten hajtson végre, amíg bizonyos feltételek teljesülnek. pl.:
  + **Elöl tesztelő ciklus (while):**

A while ciklus addig ismétli az utasításokat, amíg egy adott feltétel igaz.

int i = 0;

while (i <= 10){

print (i);

i++;

}

* + **Hátul tesztelő ciklus (do while):**

A do while ciklus először az utasításokat hajtja végre, majd utána ellenőrzi a feltételt. Ha a feltétel igaz, akkor újra végrehajtja az utasításokat.

int i = 0;

do{

print (i);

i++;

}while (i <= 10);

* + **Számláló ciklus (for):**

A for-ciklus egy iteráló változóval járja be egy adott tartományt vagy kollekciót, és minden iterációban végrehajtja az adott utasításokat.

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

print (i);

}

**Függvények, eljárások**

A függvény a programok egy viszonylag zárt, független része, amely a bemeneti adatok felhasználásával előállítja a kimenetén megjelenő értéket. Függvényeket általában akkor használunk: ha egy feladatrészt többször kell végrehajtanunk; ha olyan részfeladatot készítünk, amelyet más programban is fel akarunk használni; vagy ha ezzel rendezettebbé, átláthatóbbá tehetjük a programunkat. A függvényeknek van nevük (azonosítójuk), lehetnek paraméterei és visszaadott értéke.

A működésük hasonló a matematikában megismert függvényekhez: y = f(x), ahol x a független változó és y a függő változó (függvény érték). Programozásban a független változót paraméternek nevezzük. Egy függvény lehet paraméter nélküli és több paraméteres is.

A függvény működése:

1. A híváskor megadott aktuális paraméterek értékét elmenti a verembe.
2. A következő utasítás címét elmenti a verembe, erre a címre (a hívást követő utasításra) tér vissza a függvény befejezése után.
3. Átadódik a vezérlés.
4. A veremben tárolt értékekhez formális paramétert rendel (ezek nevével hivatkozhatunk a tárolt adatokra).
5. A függvényben megadott utasítások végrehajtása után, vagy a return utasítás hatására visszatér a híváskor elmentett címre. A visszatéréssel egyidőben a verem állapotát visszaállítja a hívást megelőzőre. Vagyis a veremben tárolt – a függvényben használt adatok – „elvesznek”.

* **Függvények:** A függvények olyan kódrészletet jelentenek, amelyek egy adott feladatot hajtanak végre. Feltételként egy adott paramétert fogadhatnak, feldolgozzák ezt a bemenetet, majd a feladat végrehajtása után egy visszatérési értéket adnak. A függvények segítenek az ismétlődő kód elkerülésében, valamint a kódbázis strukturáltabbá tételében.

public class Main {

// Egyszerű függvény, amely két számot ad össze és visszaadja az eredményt

public static int add(int a, int b) {

return a + b;

}

public static void main(String[] args) {

// Hívás a függvényre és eredményének tárolása egy változóban

int result = add(3, 5);

// Az eredmény kiíratása

System.out.println("Az összeg: " + result);

}

}

* **Eljárások:** Az eljárások hasonlóak a függvényekhez, azonban nincs visszatérési értékük. Az eljárásokat gyakran akkor használják, ha valamilyen műveletet kell végrehajtani anélkül, hogy visszatérne egy értékkel. Az eljárások akkor lehetnek hasznosak, amikor például egy adatstruktúrát kell módosítani.

public class Main {

// Egyszerű eljárás, amely üdvözli a megadott nevű személyt

public static void greet(String name) {

System.out.println("Hello, " + name + "!");

}

public static void main(String[] args) {

// Hívás az eljárásra

greet("John");

}

}

* **Alprogramok, szubrutinok:** Az alprogramok vagy szubrutinok olyan programrészek az imperatív programozási nyelvekben, amelyek feladata az adott feladatok elvégzése. Az alprogramok segítségével strukturálhatjuk és szervezhetjük a kódot, valamint újrafelhasználható, könnyen karbantartható kódstruktúrát hozhatunk létre.

**Adatátadás**

Az adatátadás pont az imperatív nyelvekben az adatok átvitelének és megosztásának módját mutatja be a függvények és eljárások között. Az adatátadás rendkívül fontos a programozásban, mert lehetővé teszi a függvények számára, hogy kapjanak bemeneti adatokat, és hogy a függvények által visszaadott eredményeket továbbítsák a többi részének a programnak. Az adatátadásnak különböző stratégiái lehetnek előnyösek különböző programozási feladatokhoz és környezetekhez. Az érték szerinti átadás például segíthet elkerülni az adatmódosítási problémákat, míg a referencia szerinti átadás hatékonyabb lehet, ha nagy mennyiségű adatot kell feldolgozni vagy manipulálni. Az adatátadás megfelelő kezelése hozzájárulhat a programok hatékonyabb és megértőbb kódjának írásához.

* **Paraméterátadás:** Abban az esetben, ha több értéket is kiszámoltatunk a függvénnyel, és azt vissza szeretnénk adni a hívó programnak, akkor ezt nem tehetjük meg függvényértékként (a függvénynek csak egy adat lehet). Ilyenkor paraméter segítségével adjuk vissza az értéket. Ez csak úgy lehetséges, ha paraméterként nem a változó értékét adjuk át, hanem annak a címét, és a függvényben az adott címre pakoljuk a kiszámított értéket.

A hívó programból az adatokat paramétereken keresztül (a veremben) adjuk át a függvénynek. A függvényben deklarált (helyi vagy lokális) változók a formális paraméterekkel együtt, szintén a veremben helyezkednek el. A műveletek eredményei is a verembe kerülnek. Mivel visszatéréskor a verem a hívást megelőző állapotra áll vissza, ezért a függvényben használt összes lokális adat elveszik.

* **Visszatérési értékek:** Az imperatív programozási nyelvekben a visszatérési értékek olyan eredmények vagy értékek, amelyeket egy függvény vagy eljárás visszaad a hívó kódnak a végrehajtása során. A visszatérési értékek lehetnek különböző típusúak, attól függően, hogy milyen nyelvet és típust használunk. A visszatérési érték típusa lehet bármilyen típusú adat, amelyet a programozó definiál vagy használ. Ez lehet egyszerű adattípus, például egész szám vagy sztring, vagy összetettebb adattípus, például tömb vagy objektum.

**Lokális és globális változók**

A lokális és globális változók az adatok érvényességének és elérhetőségének fontos fogalmait mutatja be az imperatív programozásban. Ezek a koncepciók segítenek a programozónak abban, hogy megfelelően szervezze és kezelje az adatokat a kódjában. A lokális és globális változók koncepciója fontos szerepet játszik a változók élettartamának és láthatóságának szabályozásában a programokban. A lokális változók segítenek abban, hogy a változók érvényességi tartományát a lehető legkisebbre szűkítsük, és így minimalizáljuk a potenciális hibalehetőségeket. A globális változók hasznosak lehetnek az adatok megosztásában különböző részei között a programnak, de túlzott használatuk könnyen vezethet a kód bonyolultságához és nehezen olvashatóvá válásához. A megfelelő használat és kezelés azonban lehetővé teszi a hatékony és jól strukturált programok írását.

* **Lokális változók:** A lokális változók olyan változók, amelyeket egy adott függvény vagy blokk hatóköre határoz meg. Ez azt jelenti, hogy ezek a változók csak az adott függvényben vagy blokkban érhetők el, és az érvényességük csak a függvény vagy blokk végrehajtása során tart. Amint a függvény vagy blokk végrehajtása befejeződik, a lokális változók érvényessége megszűnik, és azok értékei elvesznek.
* **Globális változók:** A globális változók olyan változók, amelyek az egész program terjedelmére kiterjednek, és bármely részének a programnak elérhetők. Ezek a változók az egész program során érvényesek, és azok értékei bármely részben módosíthatók vagy lekérdezhetők. A globális változók kinyilvánítása a főprogramban vagy azon kívül, azaz a függvények és blokkokon kívül történik.

**Verem (stack) szerepe a függvényhívásoknál**

A verem (stack) nagyon fontos szerepet játszik a függvényhívások kezelésében az imperatív programozási nyelvekben. Amikor egy függvényt meghívunk, a hívás során a program egy verem adatszerkezetben tárolja a függvényhívásokkal kapcsolatos információkat. Emellett nagyon hatékony adatszerkezet a függvényhívások kezelésére az imperatív nyelvekben, mivel kulcsfontosságú a program futása és a memóriakezelés szempontjából. Segítségével a programok könnyen nyomon követhetik a függvényhívásokat és a visszatéréseket, valamint hatékonyan kezelhetik a lokális változókat és a visszatérési címeket.

A függvény működése:

1. A híváskor megadott aktuális paraméterek értékét elmenti a verembe.
2. A következő utasítás címét elmenti a verembe, erre a címre (a hívást követő utasításra) tér vissza a függvény befejezése után.
3. Átadódik a vezérlés.
4. A veremben tárolt értékekhez formális paramétert rendel (ezek nevével hivatkozhatunk a tárolt adatokra).
5. A függvényben megadott utasítások végrehajtása után, vagy a return utasítás hatására visszatér a híváskor elmentett címre. A visszatéréssel egyidőben a verem állapotát visszaállítja a hívást megelőzőre. Vagyis a veremben tárolt – a függvényben használt adatok – „elvesznek”.