Programozás alapjai

Írásbeli vizsga – 2019. december 9.

2-es szint:

1. **Melyek a C nyelv ciklus utasításai? Adja meg a szintaktikájukat, jellemezze működésüket! (10 pont)**

Az utasítások automatikus ismétlést biztosító programszerkezet. Az ismétlés addig tart, amig az ismétlési feltétel igaz (nem nulla).

A ciklus befejezi működését, amikor a vezérlő feltétel hamissá (nulla) válik. Az a ciklus, amelynek vezérlő feltétele soha nem lesz hamis, az a **végtelen ciklus**.

Ciklusból kilépni a vezérlő feltétel hamissá válása előtt a **break** vagy a **return** utasítással lehet.

A ciklus utasításblokkjának (törzsének) vége előtt a következő iterációra lépni a **continue** utasítással lehet.

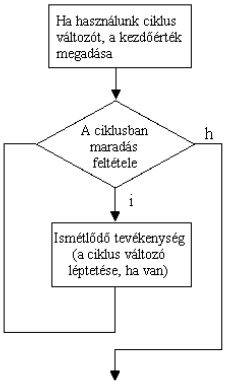
A vezérlőfeltétel-kifejezés egészre konvertálódó, logikai kifejezés kell legyen.

A vezérlőfeltétel-kifejezés kiértékelésének helye alapján lehet a ciklus:

- **Elöltesztelő ciklus:** A vezérlő feltétel az utasítás végrehajtása előtt kiértékelődik.
- **Hátultesztelő ciklus:** A vezérlő feltétel az utasítás végrehajtása után értékelődik ki (vagyis az utasítás legalább egyszer mindig végrehajtódik).

While ciklus: Addig ismétli az utasításokat (a ciklus törzsét), amíg a vezérlő feltétel (kifejezés) értéke igaz (nem nulla).

```
while (kifejezés)
        utasítás(ok)
```

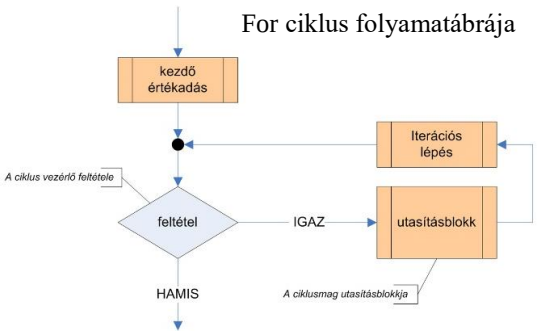


While ciklus folyamatábrája

For ciklus: Akkor használjuk, ha a ciklusmagban megadott utasítást adott számszor kívánjuk végrehajtani (az ismétlések száma adott. Elöltesztelő ciklus. A kifejezések opcionálisak ;-vel elválasztva. A léptető kifejezésben a léptető operátor prefix és postfix alakja is használható. A while ciklus speciális esete, ezért a for ciklus mindig átírható while-lá, illetve fordítva is.

```
for (inicializáló_kifejezés; feltétel_kifejezés;
léptető_kifejezés)
        utasítás(ok)
```

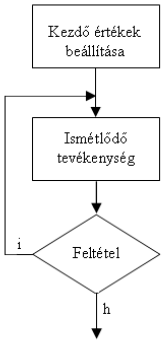
For ciklus folyamatábrája




```
inicializáló_kifejezés;
while(feltétel_kifejezés)
{
        utasítás(ok);

        léptető_kifejezés;
}
```

Do-while ciklus: Az utasítás végrehajtását követi a kifejezés kiértékelése, ezért legalább egyszer mindig végrehajtódik a ciklusmag. Hátultesztelő ciklus. Ha a kifejezés igaz (nem nulla), akkor új iteráció kezdődik, ha hamis (nulla), akkor a ciklus befejeződik.



Do-while ciklus folyamatábrája

```
do
        utasítás(ok)
while(kifejezés);
```

Végtelen ciklus: Vezérlő feltétele mindig igaz. Általában akkor használjuk, amikor nem tudunk kilépési feltétel megfogalmazni. A végtelen ciklus megszakításáról a programozónak kell gondoskodni vezérlésátadó utasítások használatával.

```
for (;;) utasítás;
while (1) utasítás;
do utasítás while (1);
```

2. Mi a tömb és hogyan tárolja a fordító a memóriában? (3 pont) Mely operátorok használhatók tömbökkel kapcsolatban és mik a használatuk szabályai? (3 pont)

Tömb: A tömb típus olyan változók halmaza, amelyek azonos típusúak és a memóriában sorfolytonosan helyezkednek el. A tömböt csak elemenként lehet kezelni. A tömbben az elemek indexelve vannak.

típus tömbnév[méret];

típus tömbnév[méret] = {vesszővel tagolt konstansok};

- **típus:** A tömbelemek típusa.
- **tömbnév:** A tömb neve.
- **[]:** Az indexelés operátora.
- **méret:** A fordító által kiszámítható konstans kifejezés, a tömbben tárolható elemek maximum száma (egész, int típusú). A méret elhagyható, ha megegyezik az inicializációs lista elemszámával.
- **index:** A hivatkozott tömbelem sorszáma (egész, int típusú).
- **{…}:** Tömb inicializálása, amely nem kötelező. Ha az inicializációs lista több elemet tartalmaz, mint a tömbméret, akkor fordítási hibát kapunk, ha kevesebbet, akkor az inicializálatlan elemek értéke 0.

3. **Írja le a függvény fogalmát! (2 pont) Mi a szerepe és hogy néz ki (szintaktika) a függvény deklarációja és a definíciója? (6 pont) Mi a függvényhívás mechanizmusa (függvényhívás lépései)? (6 pont)**

Függvény: A függvény a program olya névvel (azonosítóval) ellátott egysége (alprogramja), amely a program más részeiből annyiszor hívható meg, ahányszor szükség van a függvényben definiált tevékenység sorozatra.

Mindig definiálni kell, ami csak egyszer szerepelhet a programban. A programon belül bárhol elhelyezkedhet, kivéve egy másik függvény törzsében. Névvel rendelkezik, a programban több helyen is hivatkozhatunk rá (meghívhatjuk), azaz többször elvégezhetjük ugyanazt a részfeladatot. Van bemenete és kimenete. Bemenete az argumentumlista, kimenete a visszatérési értéke.

A függvény visszatérési értéket a **return** utasítással adjuk meg, így visszatérünk a függvény hívási helyére, ugyanakkor a return utáni utasítások már nem hajtódnak végre. A függvény visszatérési típusa a tömb kivételével bármi lehet.- Alapértelmezetten a visszatérési típus ANSI C-ben **int**.

A függvény definíciója:

<visszatérési típus> függvény neve (<paraméter deklarációs lista) → a függvény fejléce
{
    <lokális definíció(k) és deklaráció(k)>
    <utasítás(ok)>
}

A függvény hívásakor megadott aktuális paraméterlistát argumentumlistának nevezzük.

A függvény paraméterlistája az argumentumlistája között típusegyeztetés történik, ezért a paraméterek magadási sorrendje rögzített. A fordító ellenőrzi a paraméterek számának és típusának egyezését, de a nevüket nem!

A függvény **deklarációja** a függvény nevét, visszatérési értékének típusát és a paraméter(ek) típusát tartalmazza.

A függvény deklarációja:

visszatérési_típus függvénynév(típus1, típus 2, …, típus n);

Függvényhívás mechanizmusa: Olyan kifejezés, amely átadja a vezérlést és az argumentumokat a hívott függvénynek.

függvénynév(argumentumlista);

A függvény hívásakor az argumentumok sorrendben, érték szerint adódnak át a hívott függvénynek. A megfelelő paraméter az argumentum aktuális értékének másolatát veszi fel értékkét. A meghívott függvény akkor tér vissza a hívási helyre, amikor a return utasítás hajtódik végre. A return utasításban szereplő kifejezés a függvény visszatérési értéke, a visszatérési érték a függvény hívás kifejezés értéke.

Minden hívás helyén elvégzi a paraméterek ellenőrzését, ezért kell a main függvény előtt deklarálni a függvényt.

3-as szint:

1. **Írja le a top-down programtervezés lényegét! (3 pont)**

A megoldandó feladatból indulunk ki. Lépésenként bontjuk részfeladatokra, amíg végrehajtható lépésekhez jutunk.

2. **Mitől függ egy rendező eljárás hatékonysága? Az alapalgoritmusok közül melyik a leghatékonyabb és miért? (4 pont)**

Fontos, hogy a rendező algoritmus a lehetőségekhez képest gyors legyen, és minél kevesebb ideiglenes memóriát foglaljon, azaz az adatokat a saját helyükön rendezze.

Hatékonyságát a cserék száma adja meg.

Rendezés minimum kiválasztással módszerek a közvetlen kiválasztásos rendezés javított változata, melynek célja a felesleges cserék kiküszöbölése. A rendezés végén növekvő sorrendben lesznek az elemek. Az összehasonlítások száma $O(n^2)$, ami a közvetlen kiválasztás módszerrel megegyezik, ugyanakkor a cserék száma O(n). Csökkenő rendezés esetén rendezés maximum kiválasztással módszert használjuk.

3. **Mi a struktúra és hogyan tárolja a fordító a memóriában? Hogyan kérdezhető le a struktúra mérete? (3 pont)**

Összetett típus. Különböző típusú, önálló névvel rendelkező, szomszédosan elhelyezkedő elemek halmaza. A struktúra típusú változó adattagjait a deklaráció sorrendjében, a memóriában folytonosan tárolja a fordító.

struct pont → struktúra típus

{

int x, y; → struktúra adattagok (mezők), tetszőleges típusú lehet, kivéve void és függvény típus

};

struct pont p1, p2; → struktúra változók

struct pont

{

int x, y;

}p1, p2;

Adattagra hivatkozás: . (pont) és → (nyíl)

A pont operátor baloldali operandusa struktúra objektum, a jobboldali operandusa a struktúrán belüli adattag objektum.

A nyíl operátor baloldali operandusa a struktúra objektumra mutató pointer, a jobboldali operandusa a struktúrán belüli adattag objektum.

A **sizeof** operátor mindig a tényleges méretet adja.