**2-es szint**

1. ***Ciklus (iterációs) utasítások - > szintaktika, működés***

* Utasítások automatikus ismétlését biztosító programszerkezet. Az ismétlés addig tart, amíg az ismétlési feltétel igaz (nem nulla).
* A vezérlőfeltétel-kifejezés egészre konvertálódó, logikai kifejezés kell legyen.
* A vezérlőfeltétel-kifejezés kiértékelésének helye alapján lehet a ciklus: **Előltesztelő** vagy  **Hátultesztelő**
* **WHILE**
  + Addig ismétli az utasításokat (a ciklus törzsét) amíg a vezérlő feltétel (kifejezés) értéke igaz (nem nulla).
  + Elöltesztelő ciklus.
  + while (feltétel) {  
    //utasítások

//Fut amíg igaz a feltétel

}

* **FOR**
  + Akkor használjuk, ha a ciklusmagban megadott utasítást adott számszor kívánjuk végrehajtani (az ismétlések száma adott).
  + Elöltesztelő ciklus.
  + A kifejezések opcionálisak ; -vel elválasztva
  + A while ciklus speciális esete, ezért a for ciklus mindig átírható while ciklussá.
  + DE a ciklustörzsben kiadott continue utasítás hatása eltérő lehet:
  + **While:** köv. iteráció a feltétel\_kif kiértékelésével kezdődik
  + **For:** köv. iteráció megkezdése előtt kiértékelődik a léptető\_kif, majd a következő iteráció a feltétel\_kif feldolgozásával kezdődik
  + **Működése:**
    - Az iterációs lépések számlálására használt ciklusváltozó inicializálásával indul a ciklus (init\_kif kiértékelése). A kifejezés megadása opcionális, típusa int.
    - Ha a feltétel\_kif igaz (értéke nem nulla), végrehajtódik a ciklustörzs utasítása; majd kiértékelődik a léptető\_kif (ciklusváltozó léptetése) és vissza 2.
    - Ha a feltétel\_kif nincs megadva, akkor annak értékét igaznak tételezi fel a fordító.
    - Ha a feltétel\_kif hamis (értéke 0), a for utasítás befejezi működését.
  + for (i=0;i<N;i++) {  
    //Utasítások, adott számú végrehajtás  
    //Ciklusmag  
    }
* **DO-WHILE:**
  + Az utasítás végrehajtását követi a kifejezés kiértékelése, ezért legalább egyszer mindig végrehajtódik a ciklusmag (hátultesztelő ciklus). Ha a kifejezés igaz (értéke nem nulla), új iteráció kezdődik; ha hamis (értéke 0) a ciklus befejeződik.
  + do {  
    //utasítások, ciklusmag  
    }while(feltétel);
  + Üresutasítás nem jöhet létre!

1. ***Mi a tömb, hogyan kezeli a fordító?***

* A tömb összetett tipus a C nyelvben
* Azonos tipusú változók halmaza
* Amelyek a memóriában sorfolytonosan helyezkednek el
* ***Operátorai, használatuk szabályai***
  + típus tömbnév[méret] = {vesszővel tagolt konstansok};
  + [ ] : indexelés operátor
  + Egydimenziós tömb (vektor) létrehozása (deklarálása): típus tömbnév[méret];
  + Egydimenziós tömb egy elemére hivatkozás: tömbnév[index];
  + tömb elemeire hivatkozni a tömb nevét követő indexelés operátorban megadott elemsorszám (index) segítségével lehet.

1. ***Függvény fogalma***

* a program olyan névvel (azonosítóval) ellátott egysége (alprogramja), amely a program más részeiből annyiszor meghívható, ahányszor szükség van a függvényben definiált tevékenység sorozatra.
* **Mi a szerepe:** A függvények olyan kis programrészek, amelyek egy jól körülhatárolt részfeladatot hajtanak végre.
* **Szintaktika:** (alábbi sztem jó)
* **Definíciója:** <visszatérési tipus> fgv. neve(<param. list. dekl.>) {  
  <Lokális definíciók, változók>  
  <utasítások>  
  }  
  ->Saját függvényeinket mindig definiálni kell  
  ->Egy fv. definíció csak egyszer szerepelhet a pr-ban  
  ->A programon belül bárhol elhelyezkedhet, kivéve egy másik függvény törzsében.
* **Deklaráció:** A fv. deklarációja a fv. nevét, visszatérési értékének típusát és a paraméterek típusát tartalmazza. Ha a fv deklarációja a paraméterek nevét is tartalmazza, akkor ezt a fv prototípusának nevezzük. Ezt mindig a fv. hívás előtt kell elhelyezni a programban
* visszatérési\_típus fvnév(tipus1, tipus2, ...);
* **Függvényhívás lépései:**
  + A fv. hívás olyan kifejezés, amely átadja a vezérlést és az argumentumokat a hívott fv-nek. 🡪 fgvnev(argumentumlista);
  + Fv. híváskor az argumentumok sorrendben, érték szerint adódnak át a hívott fv-nek. Azaz a megfelelő paraméter az argumentum aktuális értékének másolatát veszi fel értékként (kívülről inicializálunk egy lokális változót).
  + a visszatérési érték a fv. hívás kifejezés értéke
  + Alapértelmezés szerint a fv. más forrásállományból (modulból) is elérhető tárolási osztálya extern
  + A fordító minden függvényhívás helyén elvégzi a paraméterek ellenőrzését → előtte deklarálni kell a fv-t
  + **KÖTELEZŐ SORREND:**
    - **header állományok (külső fgv. deklarációk) -> beépítése (#include)**
    - **globális definíciók**
    - **saját fgv.-ek prototipusa**
    - **main() fgv. definíciója**
    - **saját fgv.-ek definíciója**

**3-as szint: (Szelienk speciálba :­\*)**

1. **Véletlenszám generátor**

* Egész:

**int tomb[N], i;**

**srand(time(NULL));**

**printf("A tomb elemei random egesz szamokkal(1-20 kozott): \n");**

**for (i=0;i<N;i++) {**

**tomb[i] = rand()%20+1;**

**printf("%d\n",tomb[i]);**

**}**

* Valós:

**double array[N];**

**int j, lower=10, upper=20;**

**double range = upper - lower;**

**double div = RAND\_MAX /range;**

**double value;**

**printf("\nRandom VALOS szamok generalasa(10-20 kozott): \n");**

**for (j=0;j<N;j++) {**

**value = lower + (rand()/div);**

**array[j] = round(value\*100) / 100;**

**printf("%.2lf\n",array[j]);**

**}**

* A visszaadott véletlenszám 0 és RAND\_MAX konstans közötti egész érték. Véletlenszám generálása egy adott tartományon belül: rand() % (felső\_határ - alsó\_határ) + alsó\_határ
* Az srand() használatával beállíthatjuk a véletlenszám generátor magszámát. Így innentől kezdve a rand() függvény hívogatásaival mindig másik számot lehet előállítani.
* Szükséges hozzá, a <time.h> header, mivel ez tartalmazza srand() és a rand() fgv-eket, illetve idő alapján generál random számokat a gép!

1. ***Mi a struktúra:***
   1. különböző típusú, önálló névvel rendelkező, szomszédosan elhelyezkedő elemek halmaza

* **Tárolása:**
  + - A struktúra típusú változó adattagjait a deklaráció sorrendjében, a memóriában folytonosan tárolja a fordító.
* **Operátorokkal hivatkozás adattagjaira:**
  + - Adattagra hivatkozás: . (pont) és →
    - A pont operátor baloldali operandusa a 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 struktúra mezőkből (adattagokból, member) áll. Adott mezőre ponttal hivatkozunk: változó.mezőnév