Miskolci Egyetem

Programozás alapjai

Vizsgatételek

Hegedűs Ádám Imre – G1BI3
2010.12.07.

1.Tétel: A C programok általános szerkezete; konstansok és változók

- egy vagy több forrásfájl (.c)
- deklarációs (include, header) fájlok -> #include előfordító utasítás segítségével építjük a forrásállományba (.h).
Funkció-orientált módszer: valamely probléma megoldásához különböző funkciókat megvalósító programegységek készítésével jutunk el. Programegységeket függvényeknek hívjuk. A megírt függvényeket a main-nel azonos forrásfájlban vagy tetszőleges számú más forrásállományokba is tehetjük. A main függvény kijelöli a program belépési pontját. A program a main függvény indításával kezdődik. Egy és csak egy main függvény kell. Moduláris programozás: minden modul önálló fordítási egységet képez -> adatvesztés elve. Minden modul külön-külön lefordíthatók, javítás, fejlesztés esetén nem kell mindet újrafordítani. Objektumokat nevekkel látjuk el, hogy tudjunk rájuk hivatkozni. A név segítségével az objektumhoz értéket rendelhetünk és lekérdezhetjük az objektumban tárolt értéket -> változók (névvel ellátott objektum).
Változók definíciója:
<*tárolási_osztály*>*típus*<*típus…*> változónév<=kezdőérték>
Előírások -> tárolási osztályok: meghatározzák az objektum elhelyezkedését, láthatóságát, élettartamát. Ha a változót a függvényeken kívül definiáljuk, akkor az globális, függvényeken belül lokális tárolási osztályúak.
Konstansok:
1. const: a változó csak olvasható. A fordító a memóriában tárolja, így mutatók segítségével indirekt módon azok értéke megváltoztatható.
2. #define utasításával létrehozott makrók hoznak konstans értéket
3. enum: csak egész típusú (int) konstansok esetén alkalmazható. Nem a memóriában tárolja őket a fordító, ezért igazi konstansok.
A C program felépítése:
preprocesszor direktívák, programok
globális deklarációk
main()
{lokálos deklarációk
utasítások}
függvény – definíciók
A C programk alapvető szintaktikai egységei:
- Azonosítók: vesszők, számjegyes sorozata, betűvel vagy alulvonással kell kezdődnie.
- Kulcsszók: csak meghatározott célra
- Állandók: karakter, lebegőpontos stb
- Karakterláncok: idézőjelek közé zárt karaktersorozat az utolsó karakter után 0 bájt kerül
Operátorok: szimbólumok, amik megmondják, hogy kell feldolgozni az operandusokat.
Egységelválasztók: utasítás vége: {} Megjegyzések: /* */ zárt karaktersorozat

2.tétel: Elemi adattípusok, típusegyeztetések előfordulása

- short/ long: tárolási hossz
- signed/unsigned: előjelek értelmezését szabályozzák
- char: legkisebb címezhető egység
- int: mérete megegyezik a regiszter méretével, hardverfüggő, short <= int <= long
- enum: lehetséges értékei konstanshalmazokból kerülnek ki, konstansnevek felhasználásával származtatjuk.
enum azonosító {felsorolás}
A felsorolásban levő konstansok int típusúak.
Float: egyszeres pontosságú lebegőpontos számok tárolására
Double: kétszeres pontosságú
Long double: az adott számítógépes környezet legnagyobb pontosságú lebegőpontos típusát fedi le
Float <= double <= long double
Ha kétoperandusú operátor kétféle operandussal rendelkezik a fordítónak azonos típusúra kell alakítani a kettőt, vagy típuskonverziót csinálni. Szélesebbről szűkebbre információvesztés!
C alaptípusai
Integrális
Char – 1byte
Int – 4 byte
Short (int) – gépfüggő min 16bit
Long (int) – gépfüggő min 32bit

Lebegőpontos
Float – gépfüggő 4byte
Double - gépfüggő. 8byte
Long double – gépfüggő 16byte

3.Tétel: Kifejezések, operátorok, a kifejezések konverziói, C utasítás, utasításcsoport

A kifejezések operandusokból, vagy operátorok kombinációjából épülnek fel. Precedencia sorrend szükséges, hogy a műveleteket a megfelelő sorrendben hajthassa végre a program. Ezek megfelelnek a szokásos matematikai műveleteknek. Bizonyos műveletek feldolgozása során bizonyos változók is megváltozhatnak ez a mellékhatás.
A C nyelven megírt program végrehajtható része utasításokból épül fel. Ezek ciklusok, programelágazások, és vezérlésátadások szervezését teszik lehetővé. Tetszőleges kifejezés utasíts lesz, ha pontosvesszőt teszünk mögé.
{}: a logikailag összefüggő deklarációkat és utasításokat egy összetett utasításba vagy blokkba csoportosítjuk.
Összetett utasítást alkalmazunk, ha:
- több logikailag összefüggő utasítást használunk
- függvények törzseként
- definíciókor

Összetett utasítás:

```
{
        utasítás1
…
}
```

Blokk:

```
{
        deklarációk
        utasítások
}
```

4.Tétel: Vezérlési szerkezetek, elágazások, switch.

If, switch: a program kódjának feltételhez kötött végrehajtása. I*f :* if(kifejezés) utasítás
Akkor hajtódik végre ha a kifejezés értéke nem nulla.
If-else: if (kifejezés) utasítás1 else utasítás2

Else-if: egymásba ágyazott if utasításoknál az újabb if utasítás az else ágban van.

```
If (kifejezés)
        Utasítás
Else if (kifejezés)
        utasítás
Else if (kifejezés)
        utasítás
Else
        utasítás
```

Program többirányú elágaztatása így lehetséges. Ha bármely kifejezés igaz akkor a hozzákapcsolt utasítás hajtódik végre, ha egyik feltétel se teljesül akkor az utolsó else utasítással folytatódik. *A switch utasítás*: ha egy egész kifejezés értékét több konstans értékkel kell összehasonlítani.

```
Switch (kifejezés) {
        Case konstans:
                utasítás
        Case konstans:
        utasítás

        Default:
                Utasítások
        }
```

A switch utasítás először kiértékeli a kifejezést, majd annak a case-nek adja a kifejezést, amelynek az értéke megegyezik a kiértékelt kifejezés értékével. És a program innen fut tovább. De ha egyik kifejezés értékével sem egyezik meg, akkor a default-tal megjelölt utasítástól folytatódik. Ha nincs default akkor a switch záró ‘‘}’’ utáni utasítással folytatódik. Break vagy return utasítással kilépünk a switch-ből.

5.Tétel: Vezérlési szerkezetek: ciklusok

A programozási nyelveken bizonyos utasítások automatikus ismétlését biztosító programszerkezetet ciklusnak nevezzük. Az ismétlődés addig tart, amíg az ismétlési feltétel igaz.
Elöl tesztelő ciklus: a vezérlőfeltétel az utasítás végrehajtása előtt kerül feldolgozásra. Itt akkor hajtódik végre a ciklus következő iterációja, ha a feltétel igaz. A while és a for elöl tesztelő ciklus. A do legalább egyszer lefut, mert a vezérlőfeltétel ellenőrzése az utasítás végrehajtása után történik. Hátul tesztelő ciklus. A while ciklus mindaddig ismétli a hozzá tartozó utasítást, amíg a vizsgált kifejezés értéke igaz. A vizsgálat mindig megelőzi az utasítás végrehajtását.
A for utasítást akkor használjuk, ha a ciklusmagban megadott utasítást számszor akarjuk végrehajtani.
For (init_kif; felt._kif; léptető_kif;)
{ Utasítás }

A for ciklus átírható while ciklussá.

```
Init_kifejezés;
While (feltétel_kif) {
        Utasítás

        Léptető_kif;
}
```

6.Tétel: Függvények, függvényprototípus, függvényhívás mechanizmusa

A függvény a C program olyan névvel ellátott egysége, amely a program más részeiből annyiszor meghívható, ahányszor csak szükség van a függvényben definiált tevékenységsorozatra. A függvény bizonyos belső objektumainak a függvényhívás során adunk értéket. A return utasítás végrehajtásakor a meghívott függvény visszatér a hívás helyére.
A saját függvényeket mindig definiálni kell, egyszer. Ha a függvény definíciója megelőzi a hívás helyét, akkor az a függvény deklarációja is.

```
<visszatérési érték típusa> függvénynév (<paraméterlista>)
<paraméterek deklarációja>;
{
  /* a függvény törzse */
  <lokális definíciók és deklarációk>
  <utasítások>
}
```

A függvény a return utasítás feldolgozásakor ad vissza értéket, amelyet a visszatérési értére konvertál. A visszaadott érték az utasításban szereplő kifejezés értéke:
return kifejezés;

Tetszőleges számú return utasítás tehető a függvénybe. A void típussal olyan függvényt készítetünk, amely nem ad vissza értéket. Itt:
return;

A függvény deklarációja:
<a visszatérési érték típusa> függvénynév ();

A függvényekhez készített prototípus, amely a paraméterek információival bővíti deklarációit, teljes leírását tartalmazza a függvénynek. A prototípus definiálja a függvény visszatérési típusát, amennyiben eltér az int típustól. A paraméterlista és az argumentumlista összevetésével a fordító ellenőrzi a paraméterek számának és típusának összeférhetőségét. Azon függvények prototípusa, amelyek nem rendelkeznek paraméterrel, a
típus fv(void);

Függvényhívás általános alakja:
Kifejezés1 (<kifejezés2>)

Kifejezés1 a függvény neve, kifejezés2 az argumentum kifejezések vesszővel tagolt listája.

7.Tétel: Tárolási osztályok, hatáskör és tárolás

A tárolási osztály meghatározza, az objektum hol jön létre a memóriában és definiálja az objektum élettartamát. Megadhatjuk deklarációban, vagy maga a fordító határozza meg. Az élettartam a program végrehajtásának olyan időszaka, amelyben az adott változó. Vagy függvény létezik. Élettartam szempontjából három csoport van:
- globális (statikus)
- lokális (automatikus)
- dinamikus

A static vagy extern tárolási osztályúak statikusak. A statikus élettartamú azonosító számára kijelölt memóriaterület a program futásának teljes időtartama alatt megmarad. A globális változók inicializálása egyetlen egyszer megy végbe.
Blokkon belül a static tárolási osztály nélkül definiált azonosítók és a függvényen belül deklarált kapcsolódás nélküli azonosítók és függvények paraméterei automatikus élettartammal rendelkeznek
Az automatikus élettartamú azonosító memóriaterülete csak abban a blokkban létezik, amelyben az azonosítót definiáltuk. A lokális azonosítókhoz a blokkba történő minden egyes belépéskor új terület kerül lefoglalásra, ami blokkból való kilépés során megszűnik. Dinamikus élettartamúak azok a memóriaterületek, amelyeket a felhasználó könyvtári függvények segítségével lefoglal, illetve felszabadít. Az azonosítók csak az érvényességi tartományon belül látható. Az érvényességi tartomány a program azon részét jelöli ki, amelynek határain belül az adott azonosítót felhasználhatjuk az objektum elérésére. Érvényességi tartományok fajtái:
- blokk szintű (lokális): ha a blokkot záró ‘’}’‘-ot eléri, az azonosító nem lesz többé elérhető
- file szintű (globális): abban a fordítási egységben látható, amelyben a deklarációja van
- függvény szintű

Ha valamelyik függvényből olyan globális változót kívánunk használni, amelynek definícióját egy másik file tartalmazza, akkor az extern tárolási osztály felhasználásával kell az azonosítót deklarálni.
Utasításcímke: egyetlen függvény szintű érvényességi tartománnyal rendelkezik, csak a függvényen belül látható.
Kapcsolódás: a különböző érvényességi tartományban deklarált, ill. az azonos érvényességi tartományban egynél többször deklarált azonosítónevek a kapcsolódás mechanizmusának felhasználásával ugyanarra az objektumra, vagy függvényre hivatkozhatnak.
3 féle kapcsolódás:
- a belső kapcsolódású azonosítók csak egyetlen fordítási egységen belül ismertek
- a külső kapcsolódású azonosítók több fordítási egységbe is ismertek
- a kapcsolódás nélküli azonosítók nem rendelkeznek állandó memóriaterülettel.

8.Tétel: Tömbök, összevetésük a mutatókkal:

A tömb (array) típus olyan objektumok halmaza, amelyek azonos típusúak, és a memóriában folytonosan helyezkednek el. A tömb elemeinek típusa void és a függvénytípus kivételével tetszőleges típus lehet. Az elemek elérése a tömb nevét követő indexelés operátorban megadott elemsorszám (index) segítségével történik. A leggyakrabban használt tömbtípus egyetlen kiterjedéssel (dimenzióval) rendelkezik. Ez a vektor. A többdimenziós tömbök esetén az elemek tárolása soronként történik.
Egydimenziós tömbök általános alakja:
Típus tömbnév[méret];

A szögletes zárójelben a tömb méretét adjuk meg, amelynek a fordító által kiszámítható konstans kifejezésnek kell lennie. Ez adja meg a tömbben tárolható elemek számát. A tömb számára a memóriában lefoglalt memóriaterület mérete a sizeof(a) kifejezéssel pontosan lekérdezhető, míg a sizeof(a[0]) egyetlen elem méretét adja meg.

Vegyünk egy 10 elemű egész vektort.
Int a[10]

Mindegyik elemre az a[i] formában hivatkozhatunk. Legyen egy p egészre mutató pointer, majd a ‘‘címe’‘ operátor segítségével állítsuk azt az a tömb helyére.
Int *p;
P=&a[0];
Ha hivatkozunk a p mutató által kijelölt (*p) objektumra, akkor valójában az a[0] elemre hivatkozunk.
Míg a p mutató változó, addig az a egy konstans mutató, amelyet a fordító rögzít a memóriában. Ezért megengedett p esetén a p=a; és p++; művelet.

Irányítsuk a mutatót a tömb első elemére:
P=&a[a0]; vagy p=a;

A tömb i-edik elemének címe:
&a[i] &p[i] a+i p+i
A tömb 0-dik eleme:
a[0] p[0] *a *p *(a+0) *(p+0)
A tömb i-edik eleme
a[i] p[i] *(a+i) *(p+i)

9.Tétel: Mutatók és mutatóaritmetika, általános célú mutatók:

A változók többnyire az értékadás során kapnak értéket.
Objektum=érték

Az értékadás operátorok bal és jobb oldalon egyaránt szerepelhetnek kifejezések, melyek lényegileg különböznek egymástól. A baloldalon szereplő kifejezés (balérték) azt az objektumot jelöli ki a memóriában, ahova a jobb oldalon megadott kifejezés (jobbérték) értékét be kell tölteni. Az értékadás művelet, nem pedig utasítás.
Mutató pointer: speciális változó, amely más objektumok címét tárolja. A program első utasításával értéket adunk a p mutatónak a ‘‘címe’‘ operátor (&) felhasználásával. Vagyis a mutató értékként kapja a változó elemét, aminek következtében a mutató a változóra fog mutatni. A ‘‘mutatott’‘ objektumra való hivatkozáskor az indirektség operátort (*) kell megadni a mutató előtt. A *mutató kifejezés a változót helyettesíti. A C nyelv lehetővé tesz típus nélküli, általános mutató használatát is:
Int x;
void *ptr=&x;
Ez sose jelöl ki memóriaobjektumot. Ha ilyen mutatóval szeretnénk a hivatkozott objektumnak értéket adni, akkor felhasználói típuskonverzióval (cast) típust kell rendelnünk a cím mellé.
(int)ptr=5;
A legtöbb mutatót visszaadó könyvtári függvény a void típussal rendelkezik.

10.Tétel: Mutatótömbök, többszörös indirekció:

Sztringtömbök létrehozásakor választhatunk kétdimenziós tömb és a mutatótömb közül. Pl.:
Int a[5][10];
Int *b[5];
Az a[2][5] és a b[2][5] szintén egy int típusú elemet jelölnek, de csak az a egy igazi kétdimenziós tömb, amely számára a fordító 50 int típusú elem tárolására alkalmas területet foglal le a memóriában.
10 * sor +oszlop
A b 5 elemű mutatóvektor. A fordító csak 5 db mutató számára foglal helyet a definíció hatására.
De így már alkalmas lesz 5x10 egész elem tárolására:
Static int s1[10], s2[10], s3[10], s4[10], s5[10];
Int *b[5]={s1,s2,s3,s4,s5};
A static tárolási osztályt azért kell használni, mert csak a globális tömbök címe ismert a fordítás folyamán.
A mutatótömbök használatának az az előnye a kétdimenziós tömbökkel szemben, hogy míg a kétdimenziós tömb minden sora ugyanannyi elemet tartalmaz, addig a mutatótömbnél az egyes sorok mérete tetszőleges.

Többszörös indirektségű mutatók: ilyenkor a mutatók definíciójában több csillag (*) szerepel.
Int x; x egy egész típusú változó
Int *p; p egy int típusú mutató (amely int objektumra mutathat)
Int **\ q egy int* típusú mutató (amely int* objektumra, egészre mutató pointerre mutathat)

11.Tétel: Függvénypointerek és függvénypointer tömbök:

A függvénynevet a függvényhívás operátor bal oldali operandusaként megadva függvényhívás kifejezést kapunk. fakt(7) melynek értéke a függvény által visszaadott érték. Ha a függvénynevet önállóan használjuk fakt, akkor egy mutatóhoz jutunk, melynek értéke az a memóriacím, ahol a függvény kódja elhelyezkedik, típusa pedig a függvény típusa.

```
Unsigned long fakt(int); /*prototípus*/
Unsigned long fakt(int n)  /*definíció*/
{
unsigned long f=1;
for (; n>0;n--)f*=n;
return f;>

}
```

unsigned long (*fptr)(int);
így létrehoztunk egy olyan mutatót, amivel a fakt függvényre mutatunk, vagyis értékként felveheti a fakt függvény címét.
Az fptr olyan pointer, ami unsigned long visszatérési értékkel és egy int típusú paraméterrel rendelkező függvényre mutathat.
Typedef segítségével előállított faktoriális függvény típus:
Faktfv*ftpr;
Ftpr=fakt;
Az ftpr rámutat a fakt függvényre. Ezek után a fakt függvény az ftpr mutató felhasználásával indirekt módon is meghívható:
f10=(*ftpr)(10); vagy f10=ftpr(10);
Hasonló összefüggés van, mint a tömb neve és a tömbelem típusára mutató pointer között.
A C nyelvben használt, hasznos adatstruktúra a függvényre mutató pointerek tömbje, amely lehet egy vagy többdimenziós. Elsősorban olyan esetekben használjuk ezeket a tömböket, amikor valamilyen választást gyorsan kívánunk feldolgozni. Tegyük fel, hogy a program vezérlését egyszerűen menüvel kívánjuk megadni:
i. Két értek beolvasása
e. Kilépes
---
1. Különbségük kiszámolása
2. Összegük kiszámolása
3.Szorzatuk kiszámolasa
4. Hanyadosuk kiszamolasa

```
#include <stdio.h>
#include <conio.h>

/* Globális változók, közös használatra */
long int a,b,c;
char menu[]="\ni. Ket ertek beolvasasa\ne. Kilepes\n----------\n1. Kulonbseguk kiszamolasa\n2. Osszeguk kiszamolasa\n3. Szorzatuk kiszamolasa\n4. Hanyadosuk kiirasa\n";
/* A függvények prototípusa */
void beolvas(void);
void kulonbseg(void);
void osszeg(void);
void szorzat(void);
void hanyados(void);

int main (void)
```

```
{
char *pointer_menu;
pointer_menu=menu;
char ch='X';
while(ch!='e')
{
   /* A menü kiírása */
     printf("%s\n", pointer_menu);
     /* A választás feldolgozása */
     ch=getch();
     //scanf("%d", &ch);
     switch (ch) {
            case 'i':
                  beolvas();
                  break;
            case '1':
                  kulonbseg();
                  break;
            case '2':
                   osszeg();
                   break;
            case '3':
                   szorzat();
                   break;
            case '4': {
                   if (b!=(int) 0)
                       hanyados();
                   else
                       printf("\tNullaval osztani tilos es eletveszelyes!\n"); }
                   break;
           default:
                   if (ch!='e') printf("\tNem jo funkcio-billentyut nyomtal meg. Probald ujra!\n"); else
printf("\tViszont latasra!\n");
            }
                }
      return 0;
}
/* A menüpontnak megfelelõ függvény definíciója */
void beolvas(void)
{
   printf("\nKerem az elso szamot: ");
  scanf("%d",&a);
   printf("Kerem a masodik szamot: ");
  scanf("%d",&b);
}
void kulonbseg(void)
{
   printf("\t%d-%d=%d\n", a, b, a-b);
}
void osszeg(void)
{
   printf("\t%d+%d=%d\n", a, b, a+b);
}
void szorzat(void)
{
   printf("\t%d*%d=%ld\n", a, b, (long int) a*b);
}
void hanyados(void)
{    printf("\t%d/%d=%.2lf\n", a, b, (double) a/b);      }
```

12.Tétel: Struktúra adattípus, struktúra pointerek, struktúra tömbök, típusdefiníciók:

A C nyelv lehetővé teszi, hogy a nyelv meglévő típusait felhasználva újabb típusokat hozunk létre. A typedef segítségével szinonim típusokat vezettünk be. Ide tartoznak a felsorolt (enum) típusok.
A C nyelven a struktúra típus több tetszőleges típusú (kivéve a void és a függvénytípust) objektum együttese. Ezek az objektumok önálló, a struktúrán belül érvényes nevekkel rendelkeznek. Az objektumok szokásos elnevezése struktúraelem, vagy adattag. A struktúra típusú változó létrehozása logikailag két részre osztható. Először deklarálnunk kell magát a struktúra típust, melyet felhasználva változókat definiálhatunk.

```
Struct struktúra_azonosító {
        Típus1 tag1;
        Típus2 tag2;
        …
        TípusN tagN;
        };
```

A struktúraazonosító a struct kulcsszóval együtt jelöli az új felhasználói típust. Hivatkozás ps mutatóval a struktúrára:
- a ps-t ráirányítjuk az s1 struktúrára
- dinamikus memóriafoglalás (helyet foglalunk a struktúra számára, és ha már nem kell a hely felszabadítjuk)

```
ps = (struct book *) malloc (sizeof(stuct book) );
if(!ps)exit(-1);
…
free(ps)
```

A pont operátor baloldali operandusa a struktúra objektum, a jobboldali operandus pedig a struktúrán belül jelöli ki az adattag objektumot. A C nyelv egy önálló operátor – a nyíl operátor (->) – biztosít az adattag-hivatkozások elvégzésére. A nyíl operátor baloldali operandusa a struktúra objektumra mutató pointer, míg a jobboldali operandus a struktúrán belül jelöli ki az adattag objektumot. A pont és a nyíl operátorok mindkét esetben – közvetlen és indirekt hivatkozás esetén – egyaránt használhatók.
Értékadás speciális esete: egy struktúra típusú változó tartalmát egy másik struktúra típusú változónak kívánjuk megfeleltetni. Az ANSI C szabvány értelmezi a struktúra objektumra vonatkozó értékadás műveletét (=).
A struktúra definíciójában is szerepelhet kezdő értékadás. Az egyes adattagokat inicializáló konstansok vesszővel elválasztott listáját kapcsos zárójelek közé kell zárni. A struktúráknak tetszőleges típusú adattagjaik lehetnek. Ha egy struktúrában valamilyen más struktúra típus. Adattagot használunk, egymásba ágyazott struktúrát kapunk.

Struktúratömb definiálása:
Struct book lib[100];
Pont és az indexelés operátorát együtt kell használni

```
Lib[13].ar=123.23;
```

Dinamikus megoldásnál mutatótömböt kell használni:
Struct book *plib[100];

13.Tétel: Az unió adattípus, uniópointerek, unió tömbök, típusdefiníció:

Helyet takarítunk meg, ha ugyanazt a memóriaterületet több objektum közösen használja (de nem egy időben). Ez union típussal valósítható meg. Az uniót gyors és hatékony gépfüggő adatkonverziók megvalósítására használjuk. A struct típussal kapcsolatos formai megoldások union típusra is alkalmazhatók. Különbség az adattagok elhelyezkedésében van.

Struktúra deklarációja:

```
Struct név {
  int a;
  long b;
  char c[5];
} változó_neve;
```

Unió deklarációja:
```
Union név {
  int a;
  long b;
  char c[5];
};
```

A struct típus méretét az adattagok összmérete adja meg, míg az union mérete megegyezik a leghosszabb adattagjának méretével.

14.Tétel: Sztrinkezelés:

Sztring: egy olyan karaktertömb(char[]), melyben a karaktersorozat végét nulla értékű bájt (’\0’) jelzi. A C nyelv nem rendelkezik önállő stringtípussal ezért karaktertömböket használ a sztringek tárolására.
Amikor helyet foglalunk valamely sztring számára, akkor a sztring végét jelző bájtot is figyelembe kell venni.

```
Char st1[10]={’N’,’Y’,’U’,’S’,’Z’,’I’,’\0’};
Char st2[]={’N’,’Y’,’U’,’S’,’Z’,’I’,’\0’};
```

A karaktertömbök inicializálása sztring konstansok felhasználásával:
```
Char st1[10]=‘‘NYUSZI’‘;
Char st2[]=‘‘NYUSZI’‘;
```

Sztringek kezelésére szolgáló könyvtári függvények:
scanf, gets – sztring beolvasása
printf, puts –sztr. kiírása
strcpy – értékadás
strcat – hozzáfűzés
strlen – sztr. hosszának lekérdezése
strcmp – sztr.-ek összehasonlítása

Sztringkezelő függvények használata esetén a STRING.H deklarációs állományt be kell építeni a forrásprogramba.

15.Tétel:Adatfolyam, filekezelés, szabványos adatfolyamok

A c nyelv nem rednelkezik adatbeviteli és –kiviteli (I/O) utasításokkal, minden ilyen feladatot könyvtári függvényekkel kell megoldanunk. A függvényeket három csoportra oszthatjuk:
1.2. operációs rendszer lehetőségeinek felhasználásával végzi az I/O műveleteket
3. közvetlenül a hardver (konzol, bios) programozásával
Az OP RENDSZER szintjén a C program számára az adatátvitel mindig adatállomány olvasását és írását jelenti, bár a program bemenete a billentyűzet, kimenete a képernyő vagy a nyomtató. Adatátviteli egység: byte, C-ben a char típus. OP RENDSZER szintjén két megoldás: szabványos (adatfolyam stream), alacsony szintű (low-level). Hordozható C forráskódnál szabványos kell.
A stream függvények az állományokat karakterek folyamának tekinti. Amikor egy fájl a stream-függvények használatával kerül megnyitásra, a nyitott állományt egy FILE struktúrával, ill. a FILE struktúrára mutató pointerrel azonosítja a rendszer.

Lehetséges adatfolyamok:
Adatfolyam – Leíró - Periféria - A művelet iránya
stdin-0-billenytűzet-input
stdout-1-képernyő-output
stderr-2-képernyő-output
stdaux-3-COM1-in/output
stdprn-4-LPT1, PRN-output

A C nyelven az adott adatállományokat a tartalmuk alapján szöveges és bináris fájlokra osztjuk. A szöveges file sorokból épül fel, és minden sor végén a CR/LF karakterpár van. A bináris állomány bájtokból álló adathalmaz, minden karaktert a 8 bites kódja alapján értelmezünk.

Fájlkezelés lépései:
1 .a fájlt azonosító mutató definiálása
2. a fájl megnyitása
3. írás a fájlba olvasás a fájlból és pozicionálás benne
4. fájl lezárása

Fájl megnyitása:
fopen függvény hívásával
FILE *fopen (const char * filename, const char *mode);
filename: az állomány OP RENDSZER által használt nevét tartalmazó sztring
mode: fájl elérését és típusát határozza meg. sztring
‘‘r’‘ létező fájl megnyitása olvasásra
‘‘w’‘ új fájl megnyitása írásra. Ha a fájl már létezik tartalma elvész.
‘‘a’‘ fájl megnyitása hozzáírásra. Nyitás után a fájl vége lesz az aktuális fájlpozíció. Ha a fájl nem létezik, akkor az fopen létrehozza.
‘‘r+’‘ létező fájl megnyitása írásra olvasásra (update)
‘‘w+’‘ új fájl megnyitása írásra olvasásra (update) ha létezik a fájl, akkor tartalma elvész
‘‘a+’‘ fájl megnyitása a fájl végén végzett írásra és olvasásra. Ha a fájl nem létezik az fopen létrehozza

A fájl típusát meghatározó t és b betűk egyike a mód karakterek után következik a mode sztringben. Az fopen függvény visszatérési értéke FILE * típusú mutató, amely kijelöli a fájlhoz tartozó FILE struktúrát a memóriában. Ha a megadott állományt nem sikerült megnyitni, vagy ha a FILE struktúrának nem sikerült helyet foglalni a memóriában, NULL lesz a függvényérték.

Fájl lezárása:
fclose függvény hívása
a fájlhoz tartozó pufferek és más területek felszabadítása

Lezárás előtt a memóriában található átmeneti pufferek tartalmának a fájlba írása az fflush függvénnyel:
fflush (FILE*);
fclose(FILE*);

Karakter olvasása és írása:
int fgetc(FILE *stream);
int fputc(int c, FILE *stream);

Sztring olvasása és írása:
char *fgets(char *s, int n, FILE *stream);
int fputs(const char *s, FILE *stream);

Formázott adatok olvasása és írása:
int fscan (FILE *stream, const char *format, ...);
int fprint (FILE *stream, const char *format, ...);

Fájl olvasásakor az feof függvény 0 értéke jelzi a fájlvége esemény bekövetkeztét:
int feof (FILE * fp);

Bináris állományok kezelése:
- bájtonként vagy blokkonként
- bájtos kezelés:fgetc és fputc függvényekkel
- a memóriablokk fájlba írására és fájlból történő visszaolvasására az fread, illetve az fwrite függvények használhatók

size_t fread(void *ptr, size_t size, size_t n, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream);

Mindkét függvény adott size méretű adatelemekkel dolgozik, melyekből n darabot ír, vagy olvas a hívás során. Az adatokat tartalmazó memóriaterület kezdetét a ptr mutató jelöli ki. A függvények visszatérési értéke a kiírt, illetve a beolvasott elemek számát adja meg. Ha ez az érték kisebb, mint az általunk kijelölt (n), akkor fájlvége, vagy fájlhiba eseményre lehet következtetni.
Rekordok írására és olvasására is a fenti memóriablokkal működő függvényeket használjuk.

Pozicionálás a fájlban:
A fájlokban mindig 0-tól kezdődő bájtpozíciókra lehet pozícionálni. A fájl elejére való pozicionálás a rewind függvénnyel végezhető el legkényelmesebben. Tetszőleges pozíció kiválasztására az fseek, az aktuális pozíció lekérdezése pedig az ftell függvényeket használjuk.
int fseek(FILE *stream, long offset, int honnan);
long ftell(FILE *stream);

Az fseek hívásánál az offset paraméter a fájlpozíció relatív távolságát tartalmazza, a honnan paraméter által kijelölt pozícióhoz képet.
SEEK_CUR az akt. pozícióhoz képest
SEEK_END a file végéhez képest
SEEK_SET a fájl elejéhez képest

Hibák kezelésére használt függvények:
feof(fp) 0 visszatérési értékkel jelzi, hogy elértük a file végét
ferror(fp) a stream hibajelző adatát teszteli, és 0 értékkel tér vissza, ha hiba volt az utolsó fájlművelet során
clearerr(fp) a hívás alaphelyzetbe állítja a stream hiba- és fájlvége jelzőjét.

16.Tétel: C preprocesszor direktívák, makroszimbólumok és makroeljárások:

Minden C fordítóprogram fontos része az előfeldolgozó (preprocesszor). A fordítóprogram a feldolgozó által előállított szöveget dolgozza fel. A preprocesszornak szóló parancsokat a sor elején álló # karakter jelzi.
Szöveghelyettesítés: #define, - ‘‘beszédes azonosítókkal láthatjuk el a C konstansokat, kulcsszavakat, ill. a gyakran használt utasításokat és kifejezéseket. Ha a makróra többé nincs szükségünk, az #undef direktíva segítségével megsemmisítjük azt. A #define azonosító helyettesítő szöveg.
Szöveges fájl beépítése: #include,
Függő fordítás: #if

Makró hívása:
azonosító (argumentumlista)

Előredefiniált makrók:
DATE A fordítás dátumát tartalmazó sztring konstans
TIME A fordítás időpontját
FILE A forrásfájl nevét

17.Tétel: Parancssor argumentum kezelés:

A C programokban a végrehajtás a main() függvénnyel kezdődik. Csak egyetlen main() függvényünk lehet . A main() véget érése egyben a program futásának végét és az irányítás visszatérését az operációs rendszerhez is jelenti.
Mainből nem hozunk létre prototípust, ezért többféle main() is használható.
int main(int argc, char *argv[])
Legalább két paramétert támogat a main(): az argc-t és az argv-t. Ez a két változó hordozza a parancssor-argumentumok számát, illetve a rájuk mutató pointert.
Az argc egy egész típusú paraméter, melynek értéke legalább egy, mert első argumentumként mindenképpen a program neve jelenik meg. Az argv paramétert karaktermutatókból álló tömbként definiáljuk. Mindegyik mutató egy-egy parancssor-argumentumra mutat.

18.Tétel: Dinamikus memória kezelés:

Dinamikus memóriahasználat során memóriablokkot foglalhatunk le, amelyet - ha már nincs szükség rá – fel is szabadíthatunk. A memóriafoglalást és felszabadítást szabványos könyvtári függvényekkel végezhetjük. Négy dinamikus allokálásra, illetve felszabadításra használt függvény: calloc(), malloc(), free(), realloc().
Stdlib.h header fájlban találhatók meg.

void *calloc(size_t x, size_t y); - A függvény elegendő memóriát foglal le egy olyan x hosszúságú tömbnek, amely y méretű elemet tartalmaz. A calloc() függvény az allokált terület első bájtjára mutató pointert adja vissza. Ha az allokálási kérelem memória hiánya miatt nem folyósítható, a calloc() nullpointert ad vissza.

Memória felszabadítása:
void free (void *ptr);
A free() függvény a heapből size-méretű memóriaterületet allokál, és annak első bájtjára mutató pointerrel tér vissza. Ha memória hiányában nem hajtható végre az allokáció, nullpointert ad vissza.

void *realloc(void *ptr, size_t size);

A realloc() függvénnyel a korábban lefoglalt a ptr által mutatott memóriatartomány méretét változtathatjuk size méretűre. Visszatérési értékként egy mutatót kapunk, amely kívánt méretűr memóriaterületre mutat. Lehet, hogy ez nem egyezik meg a pointerként átadott mutatóval, mert a függvénynek el kellett mozgatnia az adott tartományt ahhoz, hogy méretét megnövelhesse. Ebben az esetben a régi blokk tartalma automatikusan átmásolódik az új területre, azaz nem vesztünk el adatokat. Ha ptr nullpointer, akkor a realloc() egyszerűen lefoglal egy size bájtnyi memóriaterületet és visszaad egy size méretű pointert.

19.Tétel: Láncolt listák:

Az adatok tárolását listaszerkezettel valósítjuk meg, a lista elemeit pedig dinamikus memóriafoglalással hozzuk létre a láncolt listában. A lista mérete dinamikusan növelhető, ill. csökkenthető. Az elemek beszúrásakor és tárolásakor csak néhány mutató másolását kell megtennünk.
Listaelem:

adat	mutató

A lista az adaton kívül a kapcsolati információkat tároló mutatókat is tartalmazza. Legegyszerűbb listaszerkezet a lineáris lista, melynek elemei egyirányú, egyszeres láncolással (mutatókkal) vannak összekapcsolva.

A lista azonosítására a startmutató szolgál. A lista végét nullaértékű mutatóval (NULL) jelezzük. A lista kezelése során szükség van segédváltozókra, ill. a lista kezdetét jelző start mutatóra. A lista felépítésekor minden elem esetén három dolgot kell megtennünk:
- helyfoglalás a listaelem számára,
- a listaelemben tárolt adatok feltöltése,
- a listaelem hozzáfűzése a listához

Listaelem törlése:
- az adott sorszámú elem lokalizálása a listában
- törlés elvégzése
- a törölt elem területének felszabadítása

Új elem beillesztése a listába:
- a beszúrás helyének lokalizálása a megelőző elem sorszáma alapján
- helyfoglalás az új listaelem számára
- a listaelem feltöltése adatokkal
- az elem beillesztése a sorszámmal kijelölt elem után

20.Tétel: A UNIX éa a C nyelv kapcsolata:

A C nyelv jól használható operációs rendszerek és fordítóprogramok írására. Több éven át csak az AT&T UNIX operációs rendszerrel szállított C-fordító képviselte a C nyelv definícióját.