Programozás alapjai 1. (BMEVIEEA100)

Gyakorlat anyaga az 9. oktatási héten

Hétfői csoportokban a feladat: gyakorlófeladatok megoldása a keddi nagy ZH-ra (pl. az előző anyaghoz mellékelt minta feladatsor. Sztringes példát mindenképp gyakoroljanak!) A többi csoportnak a feladat: main függvény változatai, függvénypointerek és rekurzió (qsort is).

Mondjátok meg nekik, hogy a jövő héten (10. hét) írják a 3. kis ZH-t!

Előadáson dinamikus adatszerkezetek voltak, typedef.

Új anyag

A `main` függvény változatai

Három változat:

```
int main();
int main(int db);
int main(int db,char ** lista);
```

Az argumentumok kiírása:

```
#include <stdio.h>

int main(int db,char ** lista){
    int i;
    if(db<1)printf("Lehetetlen.\n");
    else{
        printf("A program neve: %s\n",lista[0]);
        for(i=1;i<db;i++)printf("Az %d. parameter: %s\n",i,lista[i]);
    }
    return 0;
}
```

Függvénypointerek

A függvénynek átadott függvénypointer használatát korábban láthattuk a `qsort` esetében. A `qsort` paraméterként kapta az összehasonlító függvényt.

A függvények kódja a memóriában található, csakúgy, mint a változók, tehát meg lehet mondani azt a memóriacímet, ahol az adott függvény kezdődik. A C nyelvben definiálhatók olyan pointerek, melyek függvények címeit tárolják. Ezekből a pointerekből tömböt is szervezhetünk, vagy átadhatjuk függvénynek paraméterként.

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

void main(){
       double (*t[4])(double),d;
       int n;

       t[0]=sin;
       t[1]=cos;
       t[2]=exp;
       t[3]=tan;
```

```
        printf("Kerek egy valos szamot: ");
        scanf("%lg",&d);
        printf("Mit szamoljak?\n1. sin\n2. cos\n3. exp\n4. tan\n");
        scanf("%d",&n);
        if(n<1||n>4)return;
        printf("Eredmeny: %g\n",t[n-1](d));
}
```

A fenti példában létrehozunk egy olyan tömböt, melynek elemei double visszatérési értékű, egy double paraméterrel rendelkező függvények címei. Ezután feltöltjük a tömböt négy függvénnyel, melyek a math.h-ban találhatók. Természetesen saját függvényeket is használhatnánk. Az utolsó sorban meghívjuk a tömb n-1-edik elemét.

A következő példa azt mutatja be, hogyan írhatunk függvényt, amely tetszőleges, egyváltozós (valós) függvény integrálját számítja ki téglalap formulával. A `teglalap` függvény `fv` függvény integrálját számítja ki [`a`,`b`] intervallumon, az intervallumot `n` egyenlő részre osztja.

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

double teglalap(double a,double b,int n,double (*fv)(double)){
        double dx=(b-a)/n,sum=0;
        int i;
        for(i=1;i<n;i++)sum+=fv(a+dx*i);
        return (sum+0.5*(fv(a)+fv(b)))*dx;
}

void main(){
        printf("I=%14.14g\n",teglalap(0,1,100,sin));
}
```

A `teglalap` függvény negyedik paramétere tehát egy olyan függvény címe, amelynek mind a paramétere, mind a visszatérési értéke double típusú. A paraméterként átadott függvényt úgy hívjuk meg, mintha egyszerű függvény volna, nem pointer.

Rekurzió

Két formája van: az önrekurzió, amikor egy függvény önmagát hívja, és a kölcsönös rekurzió, amikor két függvény felváltva hívogatja egymást.

```
#include <stdio.h>

int faktorialis(int n){
        int v;
        if(n<2)return 1;
        v= n*faktorialis(n-1);
        return v;
}

void main(){
        printf("7 faktorialisa=%d\n",faktorialis(7));
}
```

A fenti példában a faktoriálist rekurzív módon számítjuk ki. Az n!=n*(n-1)!. Itt is, mint minden rekurziót tartalmazó programnál arra kell figyelni, hogy a rekurzió egyszer biztosan véget érjen. Jelen esetben ez akkor következik be, ha n értéke 2 alá csökken, ekkor ugyanis a függvény nem fogja magát hívni.

Hogyan működik a fenti kód? Rekurzió esetén ehelyett a következőt kell elképzelnünk:

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

```
int faktorialis1(int n1){
	return 1;
}

int faktorialis2(int n2){
	int v2;
	if(n2<2)return 1;
	v2= n2*faktorialis1(n2-1);
	return v2;
}

int faktorialis3(int n3){
	int v3;
	if(n3<2)return 1;
	v3= n3*faktorialis2(n3-1);
	return v3;
}

int faktorialis4(int n4){
	int v4;
	if(n4<2)return 1;
	v4= n4*faktorialis3(n4-1);
	return v4;
}

int faktorialis5(int n5){
	int v5;
	if(n5<2)return 1;
	v5= n5*faktorialis4(n5-1);
	return v5;
}

int faktorialis6(int n6){
	int v6;
	if(n6<2)return 1;
	v6= n6*faktorialis5(n6-1);
	return v6;
}

int faktorialis7(int n7){
	int v7;
	if(n7<2)return 1;
	v7= n7*faktorialis6(n7-1);
	return v7;
}

void main(){
	printf("7 faktorialisa=%d\n",faktorialis7(7));
}
```

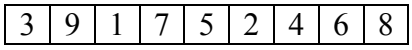
Azaz amikor egy rekurzív függvény meghívja magát, az új meghívásnál új, azonos nevű változók jönnek létre, tehát nem rontódnak el a korábbi változók értékei, de nem is használhatjuk a korábbi változók értékeit! A rekurzió használatának egyik gátja pont az, hogy minden lépésben új változók jönnek létre a verem memóriában, ami túl sok változó, vagy túl nagy hívási mélység esetén betelhet.

Saját QuickSort

A quicksort algoritmus a következőképpen működik:

1. Keressük meg a tömb középső elemét! Mivel ez nem megy az elemek végigolvasása nélkül, amire nem akarunk időt vesztegetni, válasszuk ehelyett mondjuk a két szélső elemet, és vegyük az átlagukat!
2. Rendezzük át úgy az elemeket, hogy a „középső”-nél kisebbek a tömb aljára kerüljenek, a nagyobbak pedig a tetejére.
3. Vegyük a középsőnél kisebb elemek résztömbjét, meg a nagyobb elemek résztömbjét, és ezeken ismételjük meg ismét az 1-2-3 lépéseket. A folyamat akkor ér véget, ha egy résztömbben max. egy elem marad.

Pl.:



Középső elem: (3+8)/2=5

Alsó blokk középsője: (3+5)/2=4, felső blokk középsője: (7+8)/2=7

A négy részből egy egyelemű, azt tehát nem bántjuk, a másik három középsője: 2, 6, 8

Már csak két blokk van alul, ahol egynél több elemünk van. Itt 1 és 3 lesz a középső elem. A sorrend természetesen nem változik.

A megvalósítás úgy történik, hogy miután kettéválasztottuk az elemeket, a két résztömbre meghívjuk ismét a quicksort függvényt, azaz saját magát. Ha egy függvény magát hívja, azt rekurziónak nevezzük.

```
void qs(double * t,int meret){
        if(meret<2)return;               // ha 1 vagy 0 tömbelem van, kész
        int left=0,right=meret-1;        // a két szélén kezdjük
        double med=(t[right]+t[left])/2;  // a középső elem kiválasztása
        do{
                while(t[left]<med)left++;// átlépjük a középsőnél kisebb elemeket
                while(t[right]>med)right--;// átlépjük a középsőnél nagyobb elemeket
                if(right>=left){ // Ha nem ment el egymás mellett a kép "pointer", felcseréljük az elemeket
                        double temp=t[left];t[left]=t[right];t[right]=temp;
                        left++;
                        right--;
                }
        }while(right>=left);     // Ha elment egymás mellett a két "pointer", kész a szétválogatás
        qs(t,right+1);           // Az alsó résztömbre meghívjuk a quicksortot
        qs(t+left,meret-left);   // A felső résztömbre meghívjuk a quicksortot
}
```

Ha ennél általánosabb megvalósításra van szükség, a „középső” elem lehet a résztömb első eleme is, a `right>=left` helyett pedig összehasonlító függvényt alkalmazhatunk.

A quicksort algoritmus két irányban láncolt listán is alkalmazható.

-

Házi feladat

```
X ****************************************************************************
C Függvényérték paramétersoron, függvénypointer, rekurzió
X ****************************************************************************
```

```
2. Mi lesz az fv(7) hívás eredménye?
int fv(int n) {
  if (n <= 1 ) return n;
  return fv(n - 1) + fv(n - 2);
}
```

```
3. Mit ír ki az alábbi program?
int cnt = 0;
int fv(int n) {
  cnt++; if (n <= 1) return n;
  return fv(n - 1) + fv(n - 2);
}
void main() { int a = fv(5);  printf("%d", cnt); }
```

```
4. Meghívható-e a main() függvény egy C programból?
[ ] Nem, mert nem tudjuk behelyettesíteni az argc, argv paramétereket.
[ ] Nem, mert az rekurzióhoz vezetne, ahol fennáll a végtelen ciklus veszélye.
[ ] Igen.
```

```
5. Mit lesz az "a" változó értéke?
int g (unsigned i) { printf("%d\n", i); return i ?  g(i-1) : 0; }
//...
a = g(10);
[ ] Nem lehet megmondani, mert a függvény örökké saját magát hívogatja, amíg
le nem állítják.
[ ] 9, mert i = 10-el lett meghívva a függvény, és 10 nem egyenlő nullával,
tehát 10 - 1 = 9 lesz a visszatérési érték.
[ ] 0, mert a legkülső hívás ebben a rekurzióban mindig 0-val tér vissza.
```

```
6. Mit ír ki az alábbi program?
#include <stdio.h>
void dinoszaurusz(int i){i=3*i+1;}
int main(){
   int x=30;
   dinoszaurusz(x);
   printf("%d\n",x);
   return 0;
}
```

```
7. Mit ír ki az alábbi program?
#include <stdio.h>
void dinoszaurusz(int * i){*i=3*i+1;}
int main(){
   int x=20;
   dinoszaurusz(&x);
   printf("%d\n",x);
   return 0;
}
```

```
8. A C nyelvben csak érték szerinti paraméterátadás létezik. Hogy írhatnánk
mégis olyan függvényt, amely megváltoztatja a paraméter értékét?
[ ] Pointerekkel.
[ ] Sehogy.
```

```
9. Mi történik az alábbi programban?
void k(void *t, int st, int se, void(*f)(void *)) {
   int i=0;
   for(; i < st; i++)
     f((char *)t + (i * se));
}
void j(void *k){
   int t = *((int *)k);
   printf("%d ", t);
}
void main(void){ int t[] = {23, 12, 45, 67, 89};
   k(t, sizeof(t)/sizeof(int), sizeof(int), j);
}
[ ] A program rendezi a tömböt és kiírja az elemeit. A k függvény közvetlen kivá-
lasztásos módszerrel rendezi a tömböt a j függvény pedig kiírja az elemeit.
[ ] A program kiírja a t tömb elemeit sorban a képernyőre. A k függvény egy tet-
szőleges tömb minden elemére meghívja a paraméterként kapott f függvényt.
A j függvény egy void*-ként kapott, int-re mutató pointer alapján kiírja az
adott címen lévő egész számot.
```

```
10. Mi f típusa az alábbi deklarációban?
int (*f)(const void*, const void*)
[ ] Egy olyan függvény címe, amely két mutatót vár és egy egész értéket ad vissza.
[ ] Egy olyan függvény címe, amely két mutatót vár és egy egész mutatót ad vissza.
```

```
11. Adott az alábbi függvénydeklaráció:
void qsort( void* base, int numel, int width, int (*comp)(const void*, const
void*) );
A qsort függvény hívásakor átadható-e negyedik paraméterként egy
int* compare(void*, void*);
szignatúrával rendelkező függvény?
[ ] Igen, de a program viselkedését nem tudjuk előre megmondani.
[ ] Nem.
```

```
12. Melyik függvényhívási mód helyes?
void function (int a) { }
void (*pfunction)(int) = function;
A: pfunction(2);
B: (*pfunction)(2);
[ ] Csak az A
[ ] Csak a B
[ ] Mindkettő
```

```
13. Mikor célszerű függvénypointert használni?
[ ] Ha szeretnénk elbonyolítani a kódot.
[ ] Ha szeretnénk a megvalósító algoritmust dinamikusan kiválasztani.
```

```
14. Mit ír ki a Main függvény, ha így hívjuk: Main(Int)?
#include <stdio.h>
int Int(int i){if (i) i*=Int(i-1); return i;}
void Main(int(*maIn)(int)){int x=32; printf("%d\n",maIn(x));}
```

```
15. void f(int *p, int m){*p=m;}
void main(){int a=0,b=5; f(&a,b);printf("a erteke: %d, b erteke: %d\n",a,b);}
[ ] a erteke: 0, b erteke: 0
[ ] a erteke: 5, b erteke: 5
[ ] a erteke: 0, b erteke: 5
```

---

16. Helyes-e a következő kód? char s[211]; scanf("%s",s)
[ ] %s-sel nem is lehet beolvasni adatot.
[ ] Igen, mert a %s-el olvassuk be, és a tömb mindig cím szerint adódik át.
[ ] Nem, mert nem s címét adtuk meg.

---

17. Adott az alábbi függvénydefiníció:
```
int f(int n, int a, int b) {
   if (n > 2)return f(--n, a+b, a);
   return a;
}
```
Mi az f(6, 1, 6); függvényhívás eredménye?

---

18. Adott az alábbi függvénydefiníció:
```
int f(int n, int a, int b){
   if (n > 2)return f(--n, a+b, a);
   return a;
}
```
Mi az f(4, 3, 1); függvényhívás eredménye?

---

19. Tekintse a következő függvénydefiníciót:
```
int f(unsigned i, unsigned a, unsigned b) {
    if (i > 1)return f(i-1, b, a+b);
    else return b;
}
```
Mi az f(1, 2, 2) függvényhívás eredménye?

---

20. Tekintse a következő függvénydefiníciót:
```
int f(unsigned i, unsigned s) {
    if (i > 0) return f(i-1, i+s);
    else return s;
}
```
Mi az f(2, 6) függvényhívás eredménye?

---

21. Tekintse a következő függvénydefiníciót:
```
int f(const char* s, unsigned i, unsigned n) {
    if(*s)return f(s+1, i+1, *s >= 'A' && *s <= 'Z' ? i+n : n);
    else return n;
}
```
Mi az f("npl1s4D6W", 1, 4) függvényhívás eredménye?

---

22. Melyik a helyes definiálása egy függvénypointernek, amelynek void * visszatérési értéke van, és két double-t vár paraméterként?
[ ] void * (*f)(double, double);
[ ] void * f(fouble, double);
[ ] f(double, double) void*;

---

23. Működőképes-e az alábbi függvény?

```
char *f(){char nev[30];printf("Ird be a neved!");scanf("%s",nev);return nev;}
```
[ ] Nem, mert pointert kellene visszaadnia, de tömböt ad vissza
[ ] Nem, mert a visszaadott pointer nem megfelelő helyre mutat
[ ] Igen

---

24. Mit ír ki a program?
```
int a(void){return 1;}
int b(void){return 2;}
void main(void){
  int (*fgv[2])()={a,b},i=1;
  printf ("%d\n",fgv[--i]());}
```
[ ] 1
[ ] 2

---

25. Mi okozza a fordítási idejű hibát?
```
float plus    (float a, float b) { return a+b; }
float minus   (float a, float b) { return a-b; }
float operation(float, float);
float evaluate(float a, float b, operation o) { return o(a,b); }
void main() { printf("%f", evaluate(1,2, plus)); }
```
[ ] Az evalute függvényben található o függvényhívásnál az indirekció operátort kell használni
[ ] A typedef kulcsszó hiánya az operation előtt
[ ] Az evaluate függvény hívásakor a plus elé kell a cím operátor

---

26. Mit ír ki a következő program?
```
typedef int func(const char *, ...);
void fv(char * form, func f, int v) {
  f(form, ((int)&v | 10) & 11);
}
void main(){ int i=4; fv("%d", printf, i); }
```

---

27. Milyen tömböket lehet rendezni a szabványos könyvtárból elérhető gyorsrendezéssel (qsort)?
[ ] Bármilyet, de beépített típusok esetén nem kell megadni az összehasonlítást végző függvény mutatóját.
[ ] Csak beépített típusokat tartalmazó tömböket.
[ ] Bármilyet, de meg kell adni az elemek méretét és az összehasonlítást végző függvény mutatóját.

---

28. Írható-e elvben olyan függvény, amely deklarációjában nem szerepel a '*' karakter, mégis megváltoztatja a paraméter értékét?
[ ] Nem, a C nyelvben csak érték szerinti paraméterátadás létezik, így pointert kell átadni, amihez muszáj '*'-ot használni valahol.
[ ] Igen, például sort(int t[], int n);

---

29. Mit ír ki a következő program?
```
void afunction(int *x){ *x=12; }
int main(){ int v=10; afunction(&v); printf ("%d",v); return 0; }
```
[ ] 12
[ ] 10
[ ] v változó címét

---

30. Mit ír ki a program?
```
void csere(int x, int y){ int b; b=x; x=y; y=b; }
int main(){ int a=7,b=2; csere(a,b); printf("%d\n",b); return 0; }
```

---

31. Mit csinál a következő függvény ?
```
long convert( char s[] ) {
       long result, r2;
       for( result = 0L; *s; s++) {
              if( *s >='0' && *s <='9' ) {
                     result = (r2 = result << 1) << 2;
                     result += *s -'0' + r2;
              }
              else return result;
       }
       return result;
}
```
[ ] Az atol függvényhez hasonlóan, egy decimális számjegyeket tartalmazó sztringet long értékké konvertál.

```
[ ] A result = (r2 = result << 1) << 2 kifejezés nem jól definiált, ezért a
convert függvény visszatérési értéke fordítófüggő.
[ ] Összeadja az s sztringben lévő decimális számjegyeket
--------------------------------------------------------------------------------
32. Mit ír ki a következő függvény, ha az egész 32 bites, és a negatív számok
kettes komplemens kódban vannak ábrázolva?
void f() {
       int i; i=-2;
       if( i>(-1)) printf("X"); else if(i<10) printf("Y");
       printf("%x%d\n", i+1, i-1);
}
[ ] X-1-3
[ ] Yffffffff-3
[ ] Y-1-2
--------------------------------------------------------------------------------
33. Átalakítható-e egy ciklussal megadott programrészlet úgy, hogy abban a
ciklust önhivatkozó (rekurzív) függvényhívás helyettesítse?
[ ] Igen.
[ ] Nem minden esetben.
--------------------------------------------------------------------------------
34. Mire használható a va_arg makró?
[ ] Változó paraméterszámú függvényekben az argumentum pointer inicializálásához
[ ] Ezzel léphetünk tovább a következő azonosító nélküli argumentumra
[ ] "Rendet rak" egy változó hosszúságú paraméterlistával rendelkező függvényből
való visszatérés előtt
--------------------------------------------------------------------------------
35. Mit ír ki a program?
main(int argc, char ** argv) { printf("%s",argv[2]); return 0; }
[ ] A program 2. paraméterét
[ ] A program paramétereinek számát.
[ ] A program 3. paraméterét
--------------------------------------------------------------------------------
36. Mit ír ki a program?
main(int argc, char ** argv) { printf("%d\n", argc); return 0; }
[ ] A program paramétereinek számát.
[ ] A program 3. paraméterét
[ ] A program 2. paraméterét
--------------------------------------------------------------------------------
37. Mit jelent az, ha egy globális változó statikus?
[ ] A változó nem érthető el más forrásállományokból.
[ ] A változó a program futása után is a memóriában marad és megőrzi értékét.
[ ] A változót kötelező inicializálni.
--------------------------------------------------------------------------------
38. Mit jelent az, ha egy lokális változó statikus?
[ ] A változót kötelező inicializálni.
[ ] A változó nem érthető el más forrásállományokból.
[ ] A változó a blokk befejeződése után is megőrzi értékét
--------------------------------------------------------------------------------
39. Mit ír ki a program?
#include<stdio.h>
int main(int argc, char ** argv){
  if (!strcmp(argv[0],"fritillaria"))
    printf("Szojuz T-10\n");
  return 0;
}
[ ] Azt, hogy "Szojuz T-10".
[ ] Ha a futtatható programállomány neve "fritillaria", akkor azt, hogy
"Szojuz T-10", különben semmit.
[ ] Ha első paraméterként azt adtuk meg, hogy "fritillaria", akkor azt, hogy
"Szojuz T-10", különben semmit.
--------------------------------------------------------------------------------
40. Melyik main() deklaráció helyes az alábbiak közül?
A int main();
B int main(int argc);
C int main(int argc, char *argv[]);
[ ] Csak A és C
[ ] Mindhárom helyes
[ ] Csak A
```

Példák

1. Készítsen olyan C programot, amely parancssori paraméterként kap egy egész számot, és kiír egy ennek megfelelő szorzótáblát! (A számot sztringként kapja!)
2. Írjon függvényt, amely átvesz egy int típusú elemekből álló tömböt, annak méretét, valamint egy olyan függvényt, amely egy int paramétert vár és visszatérési értéke void. A megírandó függvény járja végig a tömböt és minden elemére hívja meg a paraméterként átvett függvényt!
3. Készítsen függvényt, mely kiszámítja és visszatér S értékével, mely a következő sorozat értéke a megadott n számra: S=1*(2+3*(4+5*(... n) ...))
4. Készítse el a tetszőleges tömböt rendezni képes quicksort függvényt