Hallgatói segédlet

Nyelvi alapok

Main függvény

Általános szintaktika

```
típus main(argumentumok száma, argumentumokat tartalmazó kétdimenziós karaktertömb)
{
    utasítások
return 0;
}


int main(int argc, char* argv[]){
return 0;
}
```

Változók

Változó deklarásának szintaktikája:

```
típus név = érték
```

Példák:

```
char c = 'a'; //egybájtos karaktertípus
int i = 50;   // négybájtos egésztípus
float f = 3.24; //négybájtok tört típus
bool b = true; //logikai típus

int N[5]; //egész típusú tömb deklarálása.
int N2[10] = {0}; // egész típusú 10 elemű tömb deklarálása nulla kezdőértékkel.
```

Módosító kulcsszavak:

- unsigned : előjel nélküli
- signed: előjeles, ezt nem szoktuk külön kiírni.
- short: rövidített értéktartományú, felezi a bájtok számát.
- long: megnövelt értéktartományú, megduplázza a lefoglalt bájtok számát.

```
unsigned char c = 'a'; //egybájtos előjelnéküli karaktertípus
short int i = 50;   // kétbájtos egésztípus
long float f = 3.24; //nyolcbájtos tört típus
```

Standard be- és kimenet

Általános szintaktikája a kiírásnak (konzolra): std::cout << valtozo

Általános szintaktikája a beolvasásnak: std::cin >> valtozo

Fajták:

- std::cout -> standard kimenet, információk kiiratása
- std::cin -> standard bemenet, sima értékek beolvasása **szóköz** karakterig vagy **sorvégig (\n)**
- std::cerr ->standard hibaüzenetek kimenete, használata megegyezik a cout-tal.

```
//kiiratas
int a = 5;
std::cout << "a erteke= " << a << std::endl;


//beolvasas
int a;
std::cin >> a;


//kiiratas
std::cerr << "Runtime error!"<< std::endl;
```

Operátorok

Típus szerint két fő csoportja oszthatjuk fel őket attól függően, hogy hány változóra van szükség (pár példa látható mindkét típusra):

- unáris : !,++,--
- bináris : +,-,*,%

Aritmetikai operátorok:

- '+' : összeadás
- '-' : kivonás
- '*' : szorzás
- '/' : osztás
- '%' : maradékos osztás
- '++': inkrementálás, érték megnövelése eggyel
- '--': dekrementálás, érték csökkentése eggyel

Hozzárendelő operátorok:

- '=' : érték hozzárendelése
- '+=' : érték hozzáadása
- '-=' : érték kivonása
- '*=' : értékkel való szorzás
- '/=' : értékkel való osztás
- '%=' : értékkel való maradékos osztás
- '>>=' : bitszintű eltolás jobbra
- '<<=' : bitszintű eltolás balra
- '&=' : bitszintű és művelet
- '^=' : bitszintű kizáró vagy művelet
- '|=' : bitszintű logikai vagy művelet

Összehasonlító operátorok:

- '==' : értékegyezés
- '!=' : értékeltérőség
- '>' : nagyobb
- '<' : kisebb
- '>=' : nagyobb egyenlő
- '<=' : kisebb egyenlő

Logikai operátorok:

- ! : negáció
- && : logikai és
- || : logikai vagy
- ^ : kizáró vagy

Kondíció ellenőrző operátor: feltétel ? igaz : hamis

```
int a,b,c;

a=2;
b=7;
c = (a>b) ? a : b;

std::cout << c << '\n';
```

Méret lekérdezése: sizeof(változó)

Nevesített konstansok, makrófüggvények, direktívák

Általános szintaktika: define név érték

Példa:

```
#define N 5 //N-ként nevezzük el az ötöt
#define INT int // ilyet is lehet csinálni, hogy típusnak adunk más nevet, de ehelyett a type

#define KONSTANS // sima konstans deklarálása
//többsoros konstans
#define VALUES 1, \
               2, \
□□□3□□□
```

Makrófüggvény szintaktika: define nev() művelet

Példa:

```
#define SZORZAS(a,b) (a)*(b)
#define min(a, b) (((a) < (b)) ? (a) : (b))
```

Amennyiben lehetséges, ajánlott kerülni a használatukat, mert a hibakeresést (debugolás) igen megnehezíthetik. Minden változót erősen ajánlott **zárójelek ()** közé rakni.

Fejlécállományok beszúrása:

```
#include <iostream> //a standard fejléc beszúrása
#include "sajat.h"  // saját implementálású fejlécállomány beszúrása
```

Feltételes direktívák (fejlécállományokban alkalmazzuk:

```
#ifdef KONSTANS //ha KONSTANS már létezik
#define N   //létezzen N
#elif KONSTANS2 //különben ha KONSTANS2 létezik
    #define N2  //akkor N2 létezzen
#endif          //lezárása a direktívának

#ifndef KONSTANS //ha KONSTANS nem létezik
#define N3   //N3 létezzen
#endif           //lezárása a direktívának
```

névterek, típuselnevezések

Névtér általános szintaktika:

```
namespace NÉV {
utasítások,deklarációk
}
```

Típuselnezés szintaktika: typedef típus újtípusnév;

Példa:

```
namespace PÉLDA{
typedef long unsigned int   TELJESEGESZ;
typedef      unsigned char  KARAKTER;
}

int main()
{
PÉLDA::KARAKTER c = 'C';
return 0;
}
```

Vezérlési szerkezetek

Logikai feltétel vizsgálata

Általános szintaktikája:

```
if (feltétel)
{
igaz ág,tasitasok
}
else if (feltétel2)
{
hamis ág újabb feltétellel,utasitasok
}

else
```

```
{
hamis ág feltétel nélkül, utasításokkal
}
```

```
  int a=5;

  if (a < 5)
  {
std::cout << "a értéke kisebb mint 5\n;
  }

  else if (a > 5)
  {
std::cout << "a értéke nagyobb mint 5\n;
  }

  else {
std::cout << "a értéke megegyezik öttel\n";
  }
```

switch case elágazás

Általános szintaktikája:

```
switch(változó){
case érték1: ha változó az érték1-et tartalmazza, utasítások
case érték2: ha változó az érték2-et tartalmazza,utasítások
case érték3: ha változó az érték3-et tartalmazza,utasítások
default: alapértelmezett ág, ha a változó értéke a fentiekhze képest eltérő.
}
```

```
  int a=5;

  switch(a){
  case 1: std::cout << "a értéke 1\n;
          break;
  case 2: std::cout << "a értéke 2\n;
          break;
  case 3: std::cout << "a értéke 3\n;
          break;
  default: std::cout << "a értéke más\n;
          break;
  }
```

Ciklusok

Fajtái:

- előtesztelős: mindig először a feltétel fut le, és utána következnek az utasítások.
- hátultesztelős: a ciklusban levő utasítások legalább egyszer biztosan lefutnak, mielőtt a feltételvizsgálat megtörténik.
- számláló: a ciklus utasításait fix darab lépésben végrehajtjuk, ezt akkor alkalmazzuk, amikor előre tudjuk, hányszor lesz végrehajta. Előtesztelős szintaktika:

```
while(feltétel)
{
    utasítások
}
```

Hátultesztelős szintaktika:

```
do
{
    utasítások
}while(feltétel);
```

számláló szintaktika:

```
for (ciklusváltozó deklaráció;feltétel, ciklusváltózó művelet)
{
utasítások
}
```

Példák: Előtesztelős:

```
int N=5;

while(N != 1){
N -= 1;
}
```

Hátultesztelős:

```
int N=0;

do {
std::cin >> N
}while(N != 1);
```

Számláló:

```
for(int i=0;i<5;i++)
std::cout << i << " ";
```

Pointerek, mutatók

Általános szintaktika:

- típus *név;
- típus *név = &változó;
- típus *név = pointernév;

Példák:

```
int *a = nullptr; //sehova se mutató pointer, inicializáláskor egyik megadási mód

int N = 5;
int *b = &N; //n memóriacímére mutató pointer, a típusuk megegyezik.
int *c = b; //b-re mutató pointer, ami végeredményben N címére fog mutatni.
```

```
int N[5] = {3};
int *d = N; //N első elemére mutatunk, itt csak a tömb nevét írjuk.
int *d = &N[1] //N második tömbelemére mutatunk, mivel sima elem, ezért & jelet is írunk!

++d; //növeljük d értékét eggyel, vagyis a harmadik elemre lépünk!
d -= 2; //csökkentjük d értékét kettővel, visszalépünk N első elemére!
*d++; // a d által mutatott változó értékének növelése eggyel!
```

Dinamikus memóriafoglalás

Általános szintaktika memóriafoglalásnál:

- típus *pointer* = *(típuskényszerítés)*malloc(változótípus mérete*változó elemszáma); //inicializálatlan kezdőértékű foglalás
- típus *pointer* = *(típuskényszerítés)*calloc(változótípus mérete,változó elemszáma); //nulla kezdőértékkel feltöltött foglalás
- típus *pointer = new típus; //sima változó vagy objektum lefoglalása
- típus *pointer = new típus [elemszám]; //tömb lefoglalása

Általános szintaktika memóriafelszabadításnál:

- free(változó);
- delete változó; //sima változó esetében
- delete[] változó //tömb esetében

Példák:

```
#include <cstdlib> //ezt semmiképp se felejtsék el beinclude-olni!

int *A,*B,*C;

A = (int*)malloc(5 * sizeof(int));
B = (int*)calloc(5, sizeof(int));
C = new int[5];

//memoriafelszabaditas
free(A);
free(B);
delete[] C;
```

Függvények és eljárások

Általános szintaktika:

```
típus függvénynév (paraméterek)
{
utasítások;
visszatérési érték;
}


void eljárásnév (paraméterek)
{
utasítások;
}
```

Függvénynél mindig van return utasítás és típus a név előtt, eljárás esetében void-ot adunk meg, nincs típusa, és nincs visszatérési érték!

Példák:

```
int szorzas(int a, int n){
std::cout << "szorzas\n";
return a*n;
}

void kiirat(int n)
{
std::cout << n << std::endl;
}
```

Paraméterátadás típusai:

- érték szerinti : ebben az esetben egy új paraméterváltozóba másolódik át az érték.
- referencia szerinti: ebben az esetben az eredeti változóval dolgozunk, csak mősik elnevezést adunk neki.
- cím szerinti: pointer típusként deklarájuk a paramétert, és az eredeti változóval dolgozunk, a címét adjuk át.

Példák:

```
int szorzas(int a, int n){ //érték szerinti átadás
std::cout << "szorzas\n";
return a*n;
}

void modosit(int& n) //referencia szerinti átadás, ahonnan meghívjuk, ott az átadott változó
{
n *= 3;
std::cout << n << std::endl;
}

void modosit(int* N, int size) //cím szerinti átadás
{
for(int i=0;i<size;i++)
std::cout << N[i] << std::endl;
}
```

Parancssori argumentumok

A lefordított programkód futtatásakor lehetőség van arra is, hogy saját paramétereket adjunk át neki az alábbi módon: ./alkalmazas.exe -parameter1 -parameter2=2

Használata programkódban az alábbi módon megoldható:

```
int main(int argc, char* argv[])
{
std::cout << "Argumentumok száma: "<<argc << std::endl;
std::string parameters[argc];

for(int i=0;i<argc;i++)
{
parameters[i] = argv[i];
}

```

```
return 0;
}
```

Fájlkezelés

Az állománykezelés három fő lépésből áll:

- megnyitjuk az állományt,
- elvégezzük a szükséges műveleteket,
- bezárjuk az állományt!

Attól függően, hogy írni, vagy olvasni fogjuk, az alábbi típusokat különböztetjük meg:

- `std::ifstream` : csak olvassuk
- `std::ofstream` : csak írjuk
- `std::fstream` : egyszerre olvassuk és írjuk, igen ritkán használjuk

Megnyitás szintaktika: fájlváltozó.open(fájlnév, beállítások)

Beállítások típusai:

- `std::ios::in` : bemeneti olvasási mód
- `std::ios::out` : kimeneti írási mód, a fájl tartalma törlődik!
- `std::ios::app` : kimeneti írási mód, a fájlpozició a fájl végén van, a fájl tartalma megmarad!
- `std::ios::binary`: bináris üzemmód

Fájlbezárás szintaktika: fájlváltozó.close();

Szöveges állományok

Fájlba írás szintaktika: fájlváltozó << változó << " " ;

Fájlból olvasás szintaktika: fájlváltozó >> változó;

Bináris állományok

Fájlba írás szintaktika: fájlváltozó.write((char*)&változo,sizeof(változó);
fájlváltozó.write((char*)tömb,sizeof(tömbtípus)*méret);

Fájlból olvasás szintaktika: fájlváltozó.read((char*)&változo,sizeof(változó);
fájlváltozó.read((char*)tömb,sizeof(tömbtípus)*méret);

Fájlpozíció változtatása szintaktika: fájlváltozó.seekg(bájtpozíció,honnan);

Itt a honnan esetében három fő esetet használtunk:

- `std::ios::beg` : a fájl elejét jelenti!
- `std::ios::curr`: a fájl aktuális beolvasott bájtját jelenti!
- `std::ios::end` : a fájl végét jelenti, és visszafele léptetünk ettől számítva!

Beolvasott bájtok számának lekérdezése: fájlváltozó.tellg();

Részletes kódpéldákat az órai kódokban találnak!

Struktúrák

Általános szintaktika:

```
struct név{
típus változó1;
típus változó2;
típus változóN;
};
```