

**Pázmány Péter Katolikus Egyetem
Információs Technológiai Kar**

Bevezetés a programozásba I

9. gyakorlat

C++: alprogramok, alprogramok kommunikációja

© 2011.11.15. Giachetta Roberto
groberto@inf.elte.hu
<http://people.inf.elte.hu/groberto>

Alprogramok

Szükségessége

- A főprogram terjedelme a feladat bonyolultságával arányos
 - egy adott bonyolultságon túl a főprogram olyan méretűvé válik, hogy áttekinthetetlen lesz a programozó számára
 - előfordulhatnak benne ismétlődő szakaszok, amelyek feleslegesen növelik a kód hosszát
 - amennyiben a program egy részét egy másik programban is használni akarjuk, manuálisan kell átmásolnunk a megfelelő kódrészletet
- A megoldás erre *kódrészletek kiemelése*, elhelyezése a program más részeiben, és egyszeri hivatkozással futtatni őket
 - ezáltal csökken a főprogram hossza, és megszűnnek az ismétlődő szakaszok

Alprogramok

Működése

- A kiemelt programrészeket nevezzük *alprogram*oknak, amelyek tetszőleges részprogramot tartalmazhatnak, és egy hívással futtathatóak
 - az utasítás végrehajtásakor meghívódik az alprogram (a főprogram átadja a vezérlést az alprogramnak), és az lefut
 - az alprogram lefutását követően a vezérlés visszaadódik a főprogramnak, és az folytatja a működését
 - egy alprogramot bármennyiszer le lehet futtatni, és bármikor meg lehet hívni
 - természetesen alprogramok meghívhatnak más alprogramokat is, így elméletileg a végtelenségig bővíthető a hívások folyamata

Alprogramok

Kommunikáció

- Az alprogramok egymásnak átadhatnak értékeket (*kommunikálhatnak*) a következő módszerekkel:
 - *globális változók/konstansok*: olyan változók, amelyek a teljes programkódban érvényben vannak, nem csak egy adott programrészben belül
 - *paraméterek*: olyan változók, amelyek az alprogram meghívásakor kapnak értéket, és a teljes alprogramban érvényesek, paraméterből bármennyit adhatunk egy alprogramnak, de a meghíváskor mindegyiknek értéket kell adnunk
 - *visszatérési érték*: az alprogram által visszaszolgáltatót érték, amely visszakerül a hívás helyére, egy alprogramnak csak egy visszatérési értéke lehet

Alprogramok

Felépítése

- Az alprogramok fajtái:
 - *eljárás*: egy utasítássorozatot hajt végre
 - *függvény*: egy számítást végez el, amelynek eredménye van, az eredményt pedig visszaadja a függvény meghívójának (ez a visszatérési érték)
- Az alprogram részei:
 - *deklarációs rész*: tartalmazza az alprogram nevét, függvény esetén a visszatérési érték típusát, illetve a paraméterek listáját
 - *alprogram törzse*: a hozzátartozó utasítássorozat
- A deklaráció és a törzs lehet a program két külön részében is, de a deklarációnak mindig meg kell előznie a törzset

Alprogramok

C++-ban

- A C++-ban csak a függvények vannak implementálva, eljárások nincsenek
 - de lehet készíteni visszatérési érték nélküli függvényt is, ekkor az eljárásnak minősül
- A C++ függvény szerkezete:

<típus> *<név>*(*<paraméterek>*) { *<utasítások>* }

ahol:

- a *típus* a visszatérési érték típusa
- a *név* a függvény neve
- a *paraméterek* változók deklarációját jelenti (ugyanúgy, mint bárhol máshol a kódban)
- a *kód* a függvény törzse, a végrehajtandó utasítássorozat

Alprogramok

Használata

- A függvény neve megegyezhet bármilyen változó nevével, a zárójel különbséget tesz köztük, illetve lehet több ugyanolyan nevű függvényünk is (erről majd következő órán)
- Pl. egy egész típusú, „egy” nevű függvény, amely az egyet adja visszatérési értéként:

```
int egy(){ return 1; }
```

- A C++ programok voltaképpen kizárólag függvényekből állnak, a főprogram is egy függvény:

```
int main(){  
    ...  
    return 0;    // visszatérési érték  
}
```

Alprogramok

Visszatérési érték

- Minden függvényben (amely nem visszatérési érték nélküli), kell szerepelnie egy érték visszaadásnak, erre a célra a **return** utasítás szolgál
 - bárhol elhelyezhetjük a függvény kódjában, de az ezt követő utasítások nem kerülnek végrehajtásra, ezért általában a függvény utolsó utasítása
 - elágazások használatával előre is tehető, illetve több is elhelyezhető egy függvényben
 - a **return** után szerepelhet érték, változó, kifejezés, de a típusának meg kell egyeznie a függvény deklarációjában megadott típussal
 - a programkód lefutásakor mindig pontosan egy **return** hajtódik végre, tehát mindig egy érték adódik vissza

Alprogramok

Visszatérési érték

- A visszatérési érték nélküli függvények típusa `void`, és ekkor a `return` után nem szerepel érték
 - pl egy `skip` nevű függvény, amely nem csinál semmit, és nem ad vissza értéket:

```
void skip(){ return; }
```
 - ezekben a függvényekben nem is kötelező kiírni a `return` utasítást, ezért csak akkor írjuk ki őket, ha a függvény működését előbb meg akarjuk szakítani, minthogy a teljes függvénytörzs kódja lefusson
 - tehát az előző függvény így is írható:

```
void skip(){}
```
- A már deklarált függvényeket magában, vagy más függvényben bármikor lefuttathatjuk (előbbi esetben *rekurzióról* beszélünk)

Alprogramok

Hívása

- Lehetőségünk van utasítás formájában lefuttatni a függvényt bármely programblokkban (ez érvényes visszatérési értékkel rendelkező és nem rendelkező függvényekre is)

```
// utasítások
```

```
<függvéynév>();
```

```
// utasítások
```

- A visszatérési értékkel rendelkező függvények mindig visszaszolgáltatnak egy értéket, ezt felhasználhatjuk (pl. értékadásban, vagy kifejezés részeként):

```
<érték> = <függvéynév>();
```

```
// ahol az érték típusa megegyezik, vagy
```

```
// kompatibilis a függvény típusával
```

Alprogramok

Hívása

- Egy függvényben bármilyen már deklarált függvényt meghívhatunk, és függvény törzse állhat csupán függvényhívásokból is, pl.:

```
int fv1() { ... }  
void fv2() { ... }  
int fv3() { fv2(); fv2(); return fv1(); }
```

- A C++-ba beépített függvények meghívása is ugyanígy történhet, pl.:

```
char c1 = toupper('a');  
// a toupper függvény visszatérési értéke char  
// típusú változó  
toupper('b');  
// ekkor ez a függvény is lefut, csak nem  
// használjuk a visszatérési értékét, az elveszik
```

Alprogramok

Példa

Feladat: Írjuk ki 20×20 csillagot a képernyőre.

- használjunk egy függvényt arra, hogy kiírjon egy sorban 20 csillagot, ez nyilván visszatérési érték nélküli lesz
- a főprogramban pedig meghívjuk ezt a függvényt 20-szor

Specifikáció:

- bemenet: nincs
- kimenet: összesen 400 db csillag 20 sorban

Megoldás:

```
#include <iostream>
using namespace std;
```

Alprogramok

Példa

Megoldás:

```
void csillag(){ // csillag nevű függvény
    for (int i = 0; i < 20; i++){
        cout << "*";
    }
    cout << endl;
} // függvény vége
```

```
int main(){ // főprogram
    for (int i = 0; i < 20; i++){
        csillag(); // függvény meghívása
    }
    return 0;
} // főprogram vége
```

Alprogramok

Példa

- A fenti függvényt meg lehet úgyis valósítani, hogy adjon vissza 20 db csillagot `string` formájában, és a főprogram ezt írja ki a kimentre
 - a függvényen belül egy `string`-be beteszünk 20 csillagot, és a végén visszaadjuk annak értékét
 - ekkor a kiíratásnál kell meghívunk a függvény eredményét, amely a visszatérési értéként adott szöveget teszi ki a képernyőre

Megoldás:

```
#include <iostream>
#include <string>
using namespace std;
```

Alprogramok

Példa

Megoldás:

```
string csillag(){
    string s = "";
    for (int i = 0; i < 20; i++)
        s += "*";
        // hozzáveszünk 20 *-ot az üres szóhoz
    return s; // visszaadjuk a csillagokat
}
int main(){
    for (int i = 0; i < 10; i++)
        cout << csillag() << endl;
        // a kiírásnál hívjuk meg a függvényt
    return 0;
}
```

Alprogramok kommunikációja

Globális változók

- A programunk egyes függvényei használhatnak *globális változókat* a kommunikációra
- A globális változók olyan változók, amelyek minden programblokkon kívül vannak deklarálva (beleértve a főprogramét is), ezért értékük bárhol elérhető a programban
 - deklarációkor ugyanúgy kaphatnak kezdőértéket
- Ezek a változók szintén a deklaráció pillanatától érvényesek, ezért célszerű őket a függvények előtt deklarálni, pl. a következő szerkezetben:

```
// include, using, ...  
// globális változók  
// függvények (a főprogram az utolsó)
```
- A nem globális változókat *lokális változóknak* nevezzük

Alprogramok kommunikációja

Példa

Feladat: Adjunk össze függvény segítségével két egész számot.

- használjunk két globális változót, amelyeket összegét a függvény visszaadja
- a főprogramban végezzük a bekérést és a kiíratást

Specifikáció:

- bemenet: két egész szám (a, b)
- kimenet: a számok összege

Megoldás:

```
#include <iostream>
using namespace std;
```

Alprogramok kommunikációja

Példa

Megoldás:

```
int a, b; // globális változók

int osszeg(){ // a és b összeadása
    return a + b;
}

int main(){
    cout << "Az első szám: "; cin >> a;
    cout << "A második szám: "; cin >> b;
    cout << "A számok összege: " << osszeg()
        << endl; // osszeg kiíratása
    return 0;
}
```

Alprogramok kommunikációja

Globális változók

- De a globális változók használata veszélyes lehet, ugyanis mivel bármikor, bármely alprogramban módosíthatjuk őket, előfordulhat, hogy véletlenül egyszer megváltozik az értékük, és onnantól a programunk hibásan működik
 - nézzük az előző példát azzal a különbséggel, hogy az összeadó függvényben nem közvetlenül $a+b$ értékét adjuk vissza, hanem először elmentjük $a+b$ -t az a -ba
 - futtassuk le kétszer az összeadó függvényt

Megoldás:

...

```
int a, b; // globális változók
```

Alprogramok kommunikációja

Globális változók

Megoldás:

```
int összead(){
    a = a + b;    // a-ba adjuk össze őket
    return a;    // a-t adjuk vissza
}

int main(){
    a = 3; b = 4;
    // adjunk nekik közvetlenül értéket
    cout << összead() << endl; // kiírja: 7
    // ezen a ponton a == 7!
    cout << összead() << endl; // kiírja: 11!
    return 0;
}
```

Alprogramok kommunikációja

Paraméterek

- Keressünk olyan megoldást, ahol ezt a hibát kiküszöbölhetjük
- Megoldás: használjunk lokális változókat, és a függvény meghívásakor a változó értékét adjuk át egy, a függvényben található változónak, és a függvény dolgozzon azon a változón, így a lefutását követően a korábban átadott érték megmarad
- Ezt a módszert *paraméterátadás*nak nevezik, egy függvénynek bármennyi paramétere lehet
- A paraméterátadásban két változó vesz részt:
 - *aktuális paraméter*: amit átadunk a függvénynek a meghíváskor
 - *formális paraméter*: ami a függvényben deklarálva van, és amit használunk a függvény törzsében

Alprogramok kommunikációja

Paraméterek

- A formális paramétereket a függvény deklarációjakor adjuk meg:

```
<típus> <függvéynév>(<típus1> <név1>, <típus2>  
<név2> /*, ... további paraméterek*/){  
    <utasítások>  
}
```

- A függvény meghívásakor az aktuális paraméterek számának meg kell egyeznie a formális paraméterek számával, illetve a típusoknak kompatibilisnek kell lennie a megfelelő formális paraméter típusával:

```
<típus1> <akt. név1>;
```

```
<típus2> <akt. név2>;
```

...

```
<függvéynév>(<akt. név1>, <akt. név2> /*, ...*/);
```

Alprogramok kommunikációja

Paraméterek

- Az aktuális paraméterek neve megegyezhet a formális paraméterekével, de ettől függetlenül mindenképpen más lokális változót jelentenek (a memóriában máshol helyezkednek el)
- Módosítsuk az előbbi számösszeadást úgy, hogy globális változók helyett paramétereket használunk

Megoldás:

...

```
int osszead(int a, int b){ // formális paraméterek
    a = a + b;
    return a;
}
```

Alprogramok kommunikációja

Paraméterek

Megoldás:

```
int main(){
    int x = 3, y = 4;
    cout << osszead(x, y) << endl; // kiírja: 7
    // x és y lesznek az aktuális paraméterek
    // a-nak 3, b-nek 4 érték adódik át
    cout << osszead(x, y) << endl; // kiírja: 7
    // ismét a-nak 3, b-nek 4 érték adódik át
    return 0;
}
```

- Tulajdonképpen egy értékadás történt:

```
// meghívás: osszead(x, y)
a = x; b = y;
// függvénytörzs lefutása: a = a + b; ...
```


Alprogramok kommunikációja

Példa

Feladat: Keressük meg egy sorban hány 'a' betű található, addig olvassuk be a sorokat, amíg üres sort nem írunk be.

- készítsünk függvényt, amely megszámolja, hány 'a' betű van egy sorban (számlálás tétele), ezt hívjuk meg a főprogramban, amennyiszer szükséges
- a soronként beolvasást a `getline` függvénnyel végezzük, és használjunk előreolvasási technikát, hogy az üres sorra már ne fusson le a feldolgozás

Specifikáció:

- bemenet: szövegsorozat (*sor*)
- kimenet: az a betűk száma soronként (*szam*)

Alprogramok kommunikációja

Példa

Megoldás:

```
#include <iostream>
#include <string>
using namespace std;

int ak_szama(string s){
    int c = 0;    // számlálás tétele
    for (int i = 0; i < s.length(); i++)
        if (s[i] == 'a')
            c++;
    return c; // visszaadjuk az eredményt
}
```

Alprogramok kommunikációja

Példa

Megoldás:

```
int main(){
    string sor; int szam;
    getline(cin, sor); // előreolvasás
    while(sor.length() > 0){
        // amíg nem üres a sor
        szam = ak_szama(sor); // függvényhívás
        cout << "a-k száma: " << szam << endl;
        getline(cin, sor);
    }
    return 0;
}
```

Alprogramok kommunikációja

Példa

Feladat: Keressük meg egy 10 elemű valós számsorozat legkisebb elemét.

- tegyük a minimumkeresést függvénybe, a bekérés és kiírás legyen a főprogramban
- használjunk intelligens tömböt

Specifikáció:

- bemenet: 10 valós szám (*szamok*)
- kimenet: a számok minimuma (*min*)

Alprogramok kommunikációja

Példa

Megoldás:

```
#include <iostream>
#include <vector>
using namespace std;

float minkereses(vector<float> szamok){
    // 10 elemű valós számok tömbje a paraméter
    float min = szamok[0]; // max. ker. tétele
    for (int i = 1; i < szamok.size(); i++)
        if (szamok[i] < min)
            min = szamok[i];
    return min;
}
```

Alprogramok kommunikációja

Példa

Megoldás:

```
int main(){
    vector<float> szamok(10); // 10 elemű tömb
    for (int i = 0; i < 10; i++) {
        string num;
        stringstream conv; // segédváltozók
        cout << "A(z) " << i+1 << ". szám: ";
        // beolvasáskor ellenőrzést végzünk
        cin >> num; conv << num; conv >> szamok[i];
        if (conv.fail()) i--;
    }
    cout << "A minimum: " << minkereses(szamok);
    return 0;
}
```

Alprogramok

Feladatok

Az alábbi feladatokban függvényt kell megvalósítani. Ha másképp nem rendelkezik a feladat, a main függvényben a megvalósított függvényt kell tesztelni az érdekes esetekre.

VI. Függvények:

1. Valósítsd meg az `int kozos(int a, int b)` függvényt, ami a közös osztók számát adja vissza.
2. Valósítsd meg az `bool tokeletes(int a)` függvényt, ami visszaadja, hogy a paraméterül kapott érték tökéletes szám-e.
4. Valósítsd meg az `int max(vector<int> v)` függvényt, ami a paraméterül kapott vektor elemei közül a legnagyobbat adja vissza.

Alprogramok

Feladatok

VI. Függvények:

9. Valósítsd meg az `int hanyor(string fajlnev)` függvényt, ami a paraméterként megkapott fájlnevhez tartozó fájlt megpróbálja megnyitni, ha nem létezik a fájl, akkor -1-et ad vissza, egyébként pedig a fájlban található sorok számát.
14. (*) Valósíts meg függvényt, amely egy szöveget átalakít úgy, hogy ha több whitespace karakter (szóköz, tab, újsor) van benne egymás után, azt egyetlen szóközzé alakítja.

Alprogramok

Feladatok

VI. Függvények:

15. Valósíts meg függvényt, ami szöveges változóból egész számot próbál csinálni, visszaadja az eredményt, és visszaadja azt is, hogy zökkenőmentes volt-e az átalakítás. Ez utóbbi érték legyen 0, ha sikeres volt, különben hogy hányadik karakter (1-től indexelve) nem volt számjegy, illetve előjel megfelelő helyen.