Függvények és eljárások

Ha nagyobb feladatot kell megoldani, akkor azt általában részfeladatokra bontjuk, hogy a megoldás átláthatóbb legyen. Az egyes részfeladatokat külön alprogramként írhatjuk meg, egy adott művlet elvégzésére.

- A jól megírt függvények elrejtik a műveletek részleteit a program azon részei elől, amelyeknek nem is kell tudniuk róluk.
- Függvényeket használva az egész program világosabbá válik és a változtatások is könnyebben elvégezhetők.
- Olyan tevékenységsorozatot célszerű függvény formájában megvalósítani, amely önállóan kezelhető (kiemelhető), másrészt ismétlődő, azaz a program futása során többször szükséges a hívása.
- Ajánlott, hogy egy függvény egy műveletet valósítson meg.

A függvénynek van visszatérési értéke (az alprogram visszatéríti a … értéket, de nincs megadva, melyik változóban kell visszaadni az eredményt).
Az eljárásnak nincs visszatérési értéke (nem kell a végére return), ha mégis valamit ki kell számítani és vissza kell adnia az eredményt, akkor azt vagy *globális változóval*, vagy *cím szerint átadott paraméterrel* oldja meg.

Függvény általános alakja:

típus név (formális paraméterlista)
{
lokális (helyi) deklarációk
utasítások } ***függvény törzs***
return eredmény
}

A függvényt / eljárást a használata (meghívása) előtt deklarálni kell.

A ***formális paraméterlistában*** a paramétereket egymástól vesszővel kell elválasztani. Ha nincs paraméter, a zárójelbe nem írunk semmit, üres marad.

A visszatérési érték típusa (***típus***) bármely típusnév lehet (int, long, float, char).

Ha a függvény nem ad vissza értéket, a visszatérési érték típusa **`void`**. Az ilyen függvényeket *eljárásoknak* is nevezzük (Pascal nyelvben egészen eltérő a két alprogram leírása, ott valóban léteznek függvények és eljárások is).

A hívott függvény a meghívójának a **`return`** utasítás segítségével adhat vissza értéket. A **`return`** utasítást tetszőleges kifejezés követheti, a típusa ugyanaz kellene legyen, mint a visszatérési érték típusa.

`return` ***kifejezés;***

A függvény hívásának általános formája:

név (aktuális paraméterlista)

Példák:

1. Kétparaméteres max nevű függvény, a két egész szám közül visszatéríti a nagyobbik értékét.

```
int max(int a, int b) // a és b formális paraméterek
{ if(a>b) return a;
     else return b;
}
```

A függvény alkalmazása, használata, meghívása:

```
int y=16;
cout<<max(5, y); // itt az 5 és y a max aktuális paraméterei
```

Ebben a példában az `a` és `b` változók a `max` alprogram leírásához használt *formális paraméterek* (nem kell létezzenek a programban, csak azt a célt szolgálják, hogy le tudjuk írni a kért műveleteket).
A max függvény hívásakor az `5` és `y` *aktuális paraméterek*, az `5` egy állandó (konstans) értéke az `y` pedig a programban deklarált létező változó.

Eljárás általános alakja:

```
void eljárás_név  ([formális paraméterlista])
{  lokális változók deklarációja
   műveletek
}
```

Az eljárásokban NINCS return, ők vagy cím szerint átadott paraméterekben adják meg az eredményeket, vagy pedig kiírják képernyőre / állományba a megoldást.

Paraméterátadás

- Az egyszerű változók esetében alapértelmezett módon a paraméterátadás az aktuális paraméterek és formális paraméterek között érték szerint történik, az értékek átmásolódnak az aktuális paraméterekből a formális paraméterekbe.
- Érték szerinti átadásnál a másolaton végzett műveletek nem látszanak a hívóprogramban.
- A tömböket a C++ cím szerint adja át.

Cím szerinti paraméterátadás (&) (referencia szerint) szükséges, ha a rendezési algoritmusok számára meg kell írni a csere alprogramot

```
void csere(int &x, int &y)
{ int seged; // lokális (helyi változó), csak itt létezik
  seged=x; x=y; y=seged;
} // itt megszűnnek (meghalnak) a lokális változók és a formális paraméterek
```

Használata:

```
int v[21]={0,9,6,7,8,2};
csere(v[1], v[5]);
```

A csere alprogram végrehajtása után a tömb elemei növekvő sorrendbe lesznek rendezve.

Példa: A szamol négyparaméteres alprogram:

- az első n paramétere egy legtöbb kilenc számjegyű egész szám
- a második db paraméterben visszaadja az n számjegyeinek a számát
- a harmadik x paraméterben visszaadja az n tükörképét
- a negyedik y paraméterben a számjegyek összegét

Leírása:

```
void szamol(long n, int &db, long &x, int &y)
{ db=0; x=0; y=0;
  do{ db++; // megszámolja a számjegyeket
      x = x*10 + n % 10; // előállítja a tükörképet
      y = y + n % 10; // kiszámítja a számjegyek összegét
      n = n / 10; // kitörli az n-ből a feldolgozott számjegyet
  while(n!=0);
}
```

Meghívása:

```
int a=0, b=0;
long c, d;
cin >> c;
szamol(c, a, d, b);
cout << "szamjegyek szama = " << a << "\n";
cout << "tukorkep = " << d << "\n";
cout << "szamjegyek osszege = " << b << "\n";
```

Globális változó:

Az alprogramok leírása előtt deklarált változókat nevezzük globális változóknak, az egész program össze alprogramja "ismeri" őket.

- Amennyiben egy programnak egy vektorral vagy mátrixxal kell dolgoznia, akkor azt globális változónak is deklarálhajuk, így nem kell paraméterként átadni.
- A lokális változó vagy formális paraméter felülírja a globális változót.
- Egy for ciklusváltozóját nem deklaráljuk globálisnak, mert nagyon sok nehezen megtalálható hibához vezethet.

```
int globalis; // globális változó, innentől létezik

void minta() {
  cout << globalis;   // alprogin belül is elérhető
  globalis += 4; // módosítható is
}

int main() {
  // itt is létezik a globális változó

  globalis = 10; // Változó inicializálása (kezdőérték adás)
  minta(); // a minta alprogram meghívása
  cout << globalis; // a módosított értéket írja ki, 14-et
}
```

Statikus változók:

- csak egy alkalommal foglal helyet a változónak és ad neki kezdőértéket, a továbbiakban mindig ugyanazt a helyet használja
- meg lehet velük számolni, hogy egy függvényt pontosan hányszor hívtunk meg

```
void func()
{//"a"
  static int x = 0; // első meghíváskor jön létre, ha változik az értéke,  akkor az "megjegyződik". Ha nem lenne static, akkor "a" pontnál létrejönne, "b" pontnál megsemmisülne, mint a lokális változók
  cout << x << endl; // kiírja az x-et
  x = x + 1;
}//"b"

int main() {
  func(); // kiírja a 0-át
  func(); // kiírja az 1-et
  func(); // kiírja a 2-est
}
```