Függvények és paraméterátadás

A függvény egy olyan programblokk, melynek végrehajtását a program bármelyik olyan helyéről lehet kezdeményezni (függvény-hívás), ahol a függvény érvényes (ahová a függvény hatásköre kiterjed). Amikor a függvényt a nevére történő hivatkozással „meghívjuk”, akkor a vezérlés (az utasítások végrehajtásának menete) átadódik a függvény-hívás helyéről a függvény első utasítására. A hívó program további végrehajtása mindaddig szünetel, amíg a függvényhez tartozó kód le nem fut. A függvény végrehajtása akkor fejeződik be, ha vagy az összes utasítása végrehajtódott, vagy egy befejezését előíró (`return`) utasításhoz nem jut. Ekkor a program végrehajtása visszakerül a hívás helyére

Definíció

Egy függvényt a fejléce azonosít, és a fejléc után írt utasítás-blokk, a kapcsos zárójelek közé tett függvény-törzs tartalmazza a végrehajtható utasításait. Ez két rész – fejléc és törzs – adja a függvény definícióját.

```
visszatérési_típus függvénynév( formális paraméterlista )
{
          ...
}
```

A fejléc tartalmazza a függvény nevét és szignatúráját. A szignatúrából kiolvasható a függvény visszatérési értéke és paraméterlistája, ami együttesen azt mutatja meg, hogy milyen típusú adatokat vár és ad vissza a függvény a hívónak. A fejléc formája:

```
visszatérési_típus függvénynév( tipus1 p1, tipus2 p2, ...)
```

Egy függvényt a típusa, a neve és a paraméterváltozóinak típusai együttesen határozzák meg, ezért lehet például több eltérő funkciójú, de ugyanolyan nevű függvényt definiálni.

A fejlécet követően kapcsos zárójelek között kell elhelyezni a függvény kódját (függvény törzs). Ez egy program-blokk, amely végrehajtható utasításokat tartalmaz. Mint minden blokkban itt is definiálhatunk helyi változókat.

Deklaráció

Sokszor írunk olyan C++ kódot, ahol egy függvényt a definiálását megelőzően már deklarálni kell, azaz ki kell jelenteni, hogy a függvényre szükségünk van. A függvény-deklaráció a függvény fejléc pontosvesszővel lezárt sora.

Paraméterek

A függvény típusa és paraméterváltozóinak típusai írják le, hogy a típus hogyan kommunikál az őt hívó programmal, azaz hogyan kap attól kezdő értéket, illetve hogyan tud visszaadni eredményt. (A teljes igazsághoz hozzátartozik, hogy a különböző programrészek úgynevezett globális változókon keresztül is kommunikálhatnak, de ennek a lehetőségnek a kihasználása sok veszélyt rejt magában, ezért mi nem használjuk.)

A formális paraméterlista típus-változó párok vesszővel elválasztott felsorolása. Ezek az úgynevezett formális paraméterváltozók a függvény helyi változóinak számítanak (ugyanúgy,

mint a függvénytörzsben deklarált változók), de rendelkeznek egy különleges tulajdonsággal: a függvény hívásakor kívülről kaphatnak kezdőértéket (a hívó függvény inicializálhatja őket), illetve – bizonyos feltételek esetén – a hívó oldalon feltűntetett változóknak értéket tudnak visszaadni.

A függvény hívása a névére való hivatkozással történik:

```
függvénynév( aktuális paraméterlista )
```

Az aktuális paraméterlista kifejezések vesszővel elválasztott felsorolása. Egy ilyen kifejezés lehet a hívó program egy változója, egy konstans érték vagy egy összetett kifejezés. Az aktuális paramétereknek számban és típusban meg kell felelniük a formális paramétereknek: a két lista azonos számú paramétert tartalmaz, és az egyes paramétereknek a típusa azonos (vagy kompatibilis). Híváskor a formális paraméterváltozó megkapja a neki megfelelő aktuális paraméter értékét.

```
függvénynév( kifejezés1, kifejezés2, ... )
```

Hívás

Egy függvényt úgy hívunk meg, hogy leírjuk a nevét, majd zárójelek között felsoroljuk az aktuális paramétereit: `fuggvenynev(kifejezés1, kifejezés2, ...)`.

Ha a függvény típusa `void`, akkor ez a hívás egy önálló utasítás, ha nem, akkor egy olyan kifejezés részeként jelenik meg, ahol a függvény visszatérési típusának megfelelő értéknek kell állnia. Erre a pontra helyettesítődik be majd a függvény visszatérési értéke, amelyet a beágyazó kifejezés felhasznál.

Az aktuális paraméter lista kifejezéseknek a felsorolása. Ezek száma megegyezik a formális paraméterek számával (kivéve, ha vannak alapértelmezés szerinti értékekkel ellátott paraméterek), és a sorrendjüket annak tudatában kell meghatározni, hogy első kifejezés az első formális paraméterváltozóval, a második a másodikkal, stb. kerül majd párba. Fontos, hogy a kimenő adatokat visszaadó paraméterek esetén az aktuális paraméternek változónak kell lenni, hiszen csak ez tudja értékül kapni a visszaadott értéket.

Paraméter átadás

A függvények paraméterei alapértelmezés szerint csak az adatbevitelt szolgálják, azaz a nyelv az úgynevezett érték szerinti paraméter-átadást támogatja. Ilyenkor a formális paraméter-változó a hívott program lokális változója, amely csak annyiban függ a hívó programtól, hogy kezdőértékét a megfelelő aktuális paraméter értékéből nyeri. Bármit is tesz ezzel a paraméter-változóval a hívott program, annak új értéke nem adódik vissza a hívó oldalon szereplő aktuális paraméterhez. Ez az oka annak, hogy a hívó oldali aktuális paraméter nemcsak változó, hanem egy tetszőleges kifejezés is lehet.

Egy kicsit más a helyzet akkor, ha ez az aktuális paraméter és a formális paraméter-változó egy tömb. (Később más, a tömböktől eltérő objektumokat is használunk majd, amelyeket `new` parancs segítségével hozunk létre. Azokra is érvényesek a most elmondottak.) Ilyenkor, ha a hívott függvényben megváltoztatjuk a formális paraméterként szereplő tömb elemeit, akkor a változást a hívó függvénybe visszatérve is látjuk. A hívott függvény az aktuális paraméterként szereplő tömb elemeit meg tudja változtatni, de maga a tömb (mérete, elemi típusa) nem változhat meg.

Például egy statikus helyfoglalású tömb (`elemitípus t[10][20]`) n×m-es részének (`int n,m`) feltöltését a `tolt(t,n,m)` függvényhívással végezhetjük el, ha a függvény:

```
void tolt( elemitípus t[][20], int &n, int &m)
{
     n = …;
     m = …;
     for(int i=0; i<n; ++i){
          for(int j=0; j<m; ++j){
               t[i][j] = …
          }
     }
}
```

Ha nem akarjuk megengedni, hogy a paraméterként átadott tömb elemeit a függvény megváltoztassa, akkor írjuk a `const` kulcsszót a tömb típusa elé a formális paraméterlistában. Ilyenkor, ha a függvény törzsében kísérlet történik a tömb elemeinek megváltoztatására, akkor fordítási hibaüzenetet kapunk. Ez persze egy erősebb tiltás annál, hogy ne adódjon vissza a változás a hívó függvénybe, hiszen ezzel a formális paraméterként szereplő tömbre mindenféle változtatást megtiltunk.

A `const` kulcsszót nemcsak a tömböknél, hanem bármelyik másik formális paraméter-változónál használhatjuk, ha közölni akarjuk a fordítóval, hogy ezt a változót a hívott programban nem akarjuk megváltoztatni.

Találkozhatunk olyan helyzettel, ahol kifejezetten kívánatos lenne a hívott program paraméter-változójának értékét visszaadni a hívó programnak. Ehhez az aktuális paraméternek természetesen változónak kell lenni, de ez még (leszámítva az előbb említett eseteket) önmagában nem elég. Ilyenkor az úgynevezett hivatkozás (referencia) szerinti paraméter átadásra van szükség. Ilyenkor a formális paraméter típusa után egy & jelet kell írnunk. Ezzel deklaráljuk, hogy a formális paraméter-változó csak egy hivatkozás (egy referencia) az aktuális paraméter-változóra, azaz nem új változó, hanem az aktuális paraméter-változónak egy szimbolikus (úgynevezett „alias”) neve. Ez az oka annak, hogy a formális paraméter-változó minden változása az aktuális paraméter-változón is rögtön látszik.

Például egy dinamikus helyfoglalású tömb (`elemitípus** t`) n×m-es méretűre (`int n,m`) történő feltöltését a `tolt(t,n,m)` függvényhívással végezhetjük el, ha a függvény:

```
void tolt( elemitípus** &t, int &n, int &m)
{
     n = …;
     m = …;
     t = new elemitípus[n];
     for(int i=0; i<n; ++i){
          t[i] = new elemitípus[m];
          for(int j=0; j<m; ++j){
               t[i][j] = …
          }
     }
}
```

Visszatérési típus

Ha a függvény visszatérési típusa nem `void`, azaz van visszatérési értéke, akkor a függvény kódjában kiszámolt visszatérési értéket egy `return` utasítás után (akár kifejezés formájában) kell leírni. Ebben a `return kifejezés` alakú utasításban a kifejezés típusa a függvény visszatérési típusa vagy azzal kompatibilis, a kifejezés értéke az úgynevezett visszatérési érték. Amikor a vezérlés a `return` utasításhoz ér, akkor a függvény kódjának végrehajtása megszakad, és a vezérlés visszatérési értékkel visszatér a függvény hívásának helyére. Ilyenkor a függvény hívás egy olyan kifejezésnek tekinthető, amely a visszatérési értékkel rendelkezik. Ezért egy ilyen hívás csak a hívó oldal egy utasításába beágyazva jelenhet meg: például értékadás jobb oldalán, összetett kifejezés részeként, egy másik függvény hívásának aktuális paramétereként. Ügyeljünk arra, hogy a függvény végrehajtása (de nem feltétlenül a függvény törzse) `return` utasítással végződjön.

Összetett visszatérési értéket struktúra típussal tudunk készíteni.

Abban az esetben, ha a függvény típusa `void`, a végrehajtás végét az üres `return` utasítással jelezzük.

Program paraméterek

Programjaink `main()` függvénye ugyanolyan függvény, mint amelyekről itt szó esett. Ennek a függvénynek is lehet paramétere, de az kizárólag az alábbi módon:

```
int main(int argc, char *argv[])
```

Honnan kaphat bemenő értékeket a `main()` függvény, és hova jut el a visszatérési értéke? Azon operációs rendszerből illetve rendszerbe ahonnan indítjuk a programot. Parancssorból egy program a projekt nevével indítható, mely név után tetszés szerinti bemenő értéket (szavakat) írhatunk szóközzel elválasztva. Az `argc` paraméterváltozóból azt olvashatjuk ki, hogy a program indításának parancssorában hány bejegyzés volt (programnév + paraméterek), az `argv[i]` pedig az `i`-edik paramétert tartalmazza.

Fordítási egységek

Egy C++ program több (cpp) kirejesztésű forrásállományból, és bizonyos deklarációkat tartalmazó fejállományokból áll. A fejállományok önmagukban nem fordíthatóak, csak bemásolni tudjuk őket egy forrásállományba az `#include "fejállomány"` utasítás segítségével.

Amikor az alkalmazásunk egyetlen fordítási egységből áll (ez a `main()` függvényt tartalmazó cpp kiterjesztésű fájl), akkor annak elején deklaráljuk a `main()`-től különböző függvényeket és ezek definícióját a `main()` után tetszőleges sorrendben helyezzük el.

Ha a programunkat több állományra bontjuk, például azért, mert bizonyos függvények csoportját később is fel akarjuk használni, akkor az egy csoportba sorolt függvények deklarációját (esetleg néhány típusdefinícióval együtt) egy külön fejállományban helyezzük el, a függvény-definíciókat pedig egy ugyanilyen nevű forrásállományba tesszük. A fejállományt bemásoljuk (`#include`) a függvény-definíciókat tartalmazó forrásállományba, valamint az összes többi olyan forrásállományba is, ahol valamelyik függvényt ezek közül meg akarjuk hívni.

Kivételkezelés

Programjaink jelentős részét teszi ki a különféle hibás működések lekezelése. Egy nullával való osztás, egy nem létező állomány megnyitása olyan események, amelyekre ha nem fordítunk figyelmet, a program abortálásához, ellenőrizetlen leálláshoz vezet. Az ilyen eseteket igyekeznünk kell elkerülni, vagy ha ezt nem is lehet, akkor a hiba keletkezésekor megfelelő reakciót kell tenni. Ez lehet a program leállítása is, ha jól érthető hibaüzenet kiírásával párosul.

Az egyszerű programok esetében még megfelelő hibakezelési technikának bizonyult az elágazás. A hiba bekövetkezésének, mint feltételnek vizsgálatával olyan programágra terelhetjük a vezérlést, amelyen a hibakezelés történik. Nagyobb méretű programok esetén az a technika már az olvashatóság, az áttekinthetőség rovására megy. Egymásba ágyazott elágazások sokaságában nehéz megtalálni, melyek a hibakezelés, melyek a normális működés vezérlési vonalai. A kivételkezelés nyelvi eszközei ezen a helyzeten segítenek.

A különféle hibaesetek bekövetkezésekor lehetőségünk van a normális végrehajtás megszakítására. Ez történhet automatikusan valamilyen hibajelenség bekövetkezésekor, de a programozó utasítására is.

```
throw kivétel;
```

Ezt hívjuk a kivétel kiváltásának vagy dobásának.

A kivétel bármi lehet: egy számkonstans, egy felsorolási típus egyik eleme, egy összetett struktúrájú rekord, amelynek típusa meghatározott. Vannak előre definiált kivétel típusok is, amelyek értékeit, a kivételeket nemcsak a programozó definiálhat és dobathat el, hanem a futtató környezet is.

Kivételdobáskor megszakad az azt tartalmazó függvénynek a végrehajtása, és a vezérlés visszatér a hívó függvényhez. Ha ott nem kezeljük le, akkor a hívó függvényt hívó függvényhez, és így tovább. A lekezeletlen kivétel futási hibát, program abortálást okoz. A kivétel kezelését egy úgynevezett `try-catch` blokk segítségével végezhetjük el. A kritikus utasításokat (függvényhívásokat), ahol kivétel keletkezhet, egy úgynevezett `try{…}` blokkba kell beágyazni. A `try{…}` blokk után `catch(kivételtípus változó){…}` blokkokat lehet elhelyezni (annyit, ahány különböző kivétel típust szeretnénk figyelni). Az a kivétel, amelyik a `try{…}` blokk valamelyik utasításának végrehajtása közben keletkezik, és típusa valamelyik `catch(kivételtípus változó)` ág típusával azonos, értékül adódik a `catch` ág változójának, a vezérlés pedig ennek az ágnak a blokkjára kerül. A változóba került objektum alapján pontosan fel lehet ismerni a hiba keletkezésének okát, és a megfelelő választ lehet rá adni. Ha nem állítjuk le a program futását, akkor a vezérlés a `try-catch` blokk utáni utasításra adódik át. Megtehető az is, hogy a kivételt továbbdobjuk (`throw`). A `try-catch` blokkok egymásba is ágyazhatóak.

```
try {
   ... // utasítások, amelyek kivételt válthatnak ki
}catch(kivételtípus ex){
   if(ex==kivétel){
      ...
   }
}
```