Programozás
(GKxB_INTM114)

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

https://github.com/wajzy/GKxB_INTM114.git

2024. február 18.

minmax1.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    cout << "Adjon meg nem negativ egesz szamokat, megkeressuk kozottuk a "
6         << "minimalisat es a maximalisat.\nKilepes negativ szam megadasaval.\n";
7    int db=0, akt=1; // inicializacio
8    int min, max;
9    while(akt >= 0) {
10     cout << "Kovetkezo szam: ";
11     cin >> akt;
12     if(akt >= 0) {
13       if(db == 0) min = max = akt; // tobbszoros hozzarendeles
14       else if(akt > max) max = akt;
15       else if(akt < min) min = akt;
16       db++; // noveles eggyel
17     }
18   }
19   if(db > 0) cout <<   "A minimum: " << min << "\nA maximum: " << max << '\n';
20   else cout << "Nem adott meg adatokat.\n";
21   return 0;
22 }
```

Változó

deklaráció típus és azonosító megadása, helye: felhasználáskor vagy előtte

definíció deklaráció + memóriaterület foglalása

inicializáció definíciókor kezdőérték megadása, pl. `int db=0;`

= (hozzárendelés) operátor

- asszociativitás: jobbról balra
- `min = max = akt;` ≡ `max = akt; min = max;`

Többirányú elágazás: *if(...) ... else if(...) ... else if(...) ... else ...*

Növelő és csökkentő operátorok

++ növelés eggyel

−− csökkentés eggyel

Létezik elő- és utótag (prefix/postfix) alak is → műveleti sorrend!

Az előtag/utótag operátorok hatása az eredményre

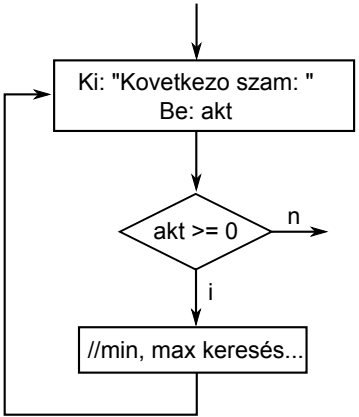
```
int a, b; // a és b értéke definiálatlan
b = 6;    // b értéke mostantól 6
a = ++b;  // 1) b értéke nő 7-re,
          // 2) ezt hozzárendeli a-hoz
a = b++;  // 1) hozzárendeli b értékét a-hoz,
          // 2) növeli b értékét 8-ra
```

minmax2.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    cout << "Adjon meg nem negativ egesz szamokat, megkeressuk kozottuk a "
6         << "minimalisat es a maximalisat.\nKilepes negativ szam megadasaval.\n";
7    int db=0, akt; // akt-nak nem kell kezdoertek
8    int min, max;
9    cout << "Kovetkezo szam: "; // kod elso elofordulasa
10   cin >> akt;
11   min = max = akt;
12   while(akt >= 0) {
13     if(akt > max) max = akt; // innen viszont eltuntek feltetelek
14     else if(akt < min) min = akt;
15     db++;
16     cout << "Kovetkezo szam: "; // kod masodik elofordulasa
17     cin >> akt;
18   }
19   if(db > 0) cout <<   "A minimum: " << min << "\nA maximum: " << max << '\n';
20   else cout << "Nem adott meg adatokat.\n";
21   return 0;
22 }
```

minmax3.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    cout << "Adjon meg nem negativ egesz szamokat, megkeressuk kozottuk a "
6         << "minimalisat es a maximalisat.\nKilepes negativ szam megadasaval.\n";
7    int db=0, akt;
8    int min, max;
9    while(cout<<"Kovetkezo szam: ", cin>>akt, akt>=0) { // , operator
10     if(!db) min = max = akt; // ! operator
11     else if(akt > max) max = akt;
12     else if(akt < min) min = akt;
13     db++;
14   }
15   // logikai kifejezes
16   if(db) cout <<   "A minimum: " << min << "\nA maximum: " << max << '\n';
17   else cout << "Nem adott meg adatokat.\n";
18   return 0;
19 }
```



Vessző operátor

- összetett, külön-külön is értelmes kifejezésekből álló kifejezés szerepeltethető ott, ahol csak egy kifejezés állhat
- a kifejezés értéke az utolsó részkifejezés értéke

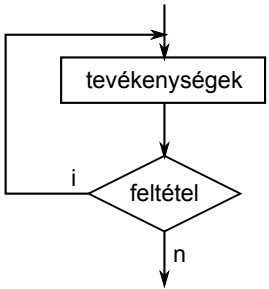
Logikai kifejezések

- `bool` típus
- `false` $\equiv 0$
- `true` $\equiv 1$
- nulla értékű egész $\rightarrow$ hamis
- nem nulla értékű egész $\rightarrow$ igaz
- `if(db) ...` $\equiv$ `if(db != 0) ...`
- `if(!db) ...` $\equiv$ `if(db == 0) ...`

haromszog1.cpp

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    int a, b, c;
5    bool megszerkesztheto = false;
6    cout << "Adja meg egy haromszog oldalhosszait!\n";
7    do {
8      do { // hatultesztelo ciklus eleje...
9        cout << "A oldal hossza: ";
10       cin >> a;
11     } while(a <= 0); // ...es vege
12     do {
13       cout << "B oldal hossza: ";
14       cin >> b;
15     } while(b <= 0);
16     do {
17       cout << "C oldal hossza: ";
18       cin >> c;
19     } while(c <= 0);
20     if(a+b<=c or b+c<=a or c+a<=b) // alternativ szintakszis
21       cout << "Ez nem szerkesztheto meg!\n";
22     else {
23       megszerkesztheto = true;
24       cout << "Megszerkesztheto.\n"; }
25   } while(not megszerkesztheto);
26   return 0; }
```

Hátultesztelő ciklus – a ciklusmag egyszer biztosan lefut




```
do {
    tevékenységek
} while(feltétel_kifejezése);
```

Logikai operátorok

- `!`, `not`: logikai nem, tagadás
- `||`, `or`: logikai (megengedő) vagy
- `&&`, `and`: logikai és

Igazságtáblázat

a	b	not a	a or b	a and b
false	false	true	false	false
false	true	true	true	false
true	false	false	true	false
true	true	false	true	true

(Rész)kifejezések kiértékelésének optimalizálása (short-circuit evaluation)

haromszog2.cpp – A feladat átfogalmazása egyszerűsíti a megoldást

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    int a, b, c;
6    cout << "Adja meg egy haromszog oldalhosszait novekvo sorrendben!\n";
7    do {
8      cout << "A oldal hossza: ";
9      cin >> a;
10   } while(a <= 0);
11   do {
12     cout << "B oldal hossza: ";
13     cin >> b;
14   } while(b < a);
15   do {
16     cout << "C oldal hossza: ";
17     cin >> c;
18   } while(c<b or a+b<=c);
19   return 0;
20 }
```

kor1.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    int sor = -5; // A kor sugara 5
6    while(sor <= 5) {
7      int oszlop = -5;
8      while(oszlop <= 5) {
9        if(5*5 >= sor*sor + oszlop*oszlop) cout << '*';
10       else cout << ' ';
11       oszlop++;
12     }
13     sor++;
14     cout << '\n';
15   }
16   return 0;
17 }
```

Problémák:

- a kurzor pozicionálása korlátozott
- rengeteg helyen szerepel ugyanaz a konstans: nehézkes módosítás, hibalehetőségek
- a karakterek kb. kétszer magasabbak, mint amilyen szélesek

Kimenet

```
    *
  *******
 *********
 *********
 *********
***********
 *********
 *********
 *********
  *******
    *
```

kor2.cpp

```
1  #include <iostream>
2  #define R 10 // A kor sugara
3  using namespace std;
4
5  int main() {
6    int sor = -R;
7    while(sor <= R) {
8      int oszlop = -R;
9      while(oszlop <= R) {
10       if(R*R >= sor*sor + oszlop*oszlop) cout << '*';
11       else cout << ' ';
12       oszlop++;
13     }
14     sor += 2; // Noveles kettovel
15     cout << '\n';
16   }
17   return 0;
18 }
```

`#define`

- szimbolikus állandók, egyszerű makrók
- előfeldolgozó „egyszerű” szöveghelyettesítést végez
- Nincs pontosvessző a végén!

Összevont operátorok

- `sor += 2;` ≡ `sor = sor+2;`
- `+=`, `-=`, `*=`, `/=`, `%=`

Egyoperandusos + és − operátorok

Kimenet

```
          *
        **********
      **********
    **********
  **********
  **********
**********
  **********
  **********
    **********
      **********
        **********
          *
```

szamlalo.cpp – Betűk, szavak, sorok számlálása

```
1  #include <iostream>
2  #include <cstdio>
3  using namespace std;
4
5  int main(void) {
6    int k, sorDb, szoDb, karDb;
7    bool szoban = false;
8    cout << "A bemenet karaktereinek, sorainak és szavainak leszámlálása\n"
9         << "A bemenet vége: Ctrl+D vagy EOF.\n\n";
10   sorDb = szoDb = karDb = 0;
11   while((k=cin.get()) != EOF) {
12     ++karDb;
13     if(k == '\n') ++sorDb;
14     if(k==' ' or k=='\n' or k=='\t') szoban = false;
15     else if(not szoban) {
16       szoban = true;
17       ++szoDb;
18     }
19   }
20   cout << "sor = " << sorDb << ", szo = " << szoDb << ", karakter = " << karDb << endl;
21   return 0;
22 }
```

Egy karakter beolvasása: `int get()`
Karakter ábrázolása `int` típusban
Bemenet vége: `EOF` → `<cstdio>`
Műveleti sorrend! `while((k=cin.get()) != EOF) {`
A `szoban` szerepe

Operátorok precedenciája és asszociativitása

Operátor	Asszociativitás
a++ a−−	balról jobbra
++a −−a +a −a ! sizeof	jobbról balra
a*b a/b a%b a+b a−b < <= > >= == != && ||	balról jobbra
= += −= *= /= %=	jobbról balra
,	balról jobbra

ordinal1.cpp – Angol sorszámnevek

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    cout << "Szam: ";
6    int szam;
7    cin >> szam;
8    if(szam == 0) cout << '0';
9    else {
10     cout << szam;
11     if(szam>10 and szam<21) cout << "th";
12     else if(szam%10 == 1) cout << "st";
13     else if(szam%10 == 2) cout << "nd";
14     else if(szam%10 == 3) cout << "rd";
15     else cout << "th";
16   }
17 }
```

Problémák:

- nagyon sok irányú elágazás
- felesleges osztások

ordinal2.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    cout << "Szam: ";
6    int szam;
7    cin >> szam;
8    if(szam == 0) cout << '0';
9    else {
10     cout << szam;
11     if(szam>10 and szam<21) cout << "th";
12     else switch(szam%10) {
13       case 1: cout << "st"; break;
14       case 2: cout << "nd"; break;
15       case 3: cout << "rd"; break;
16       default: cout << "th";
17     }
18   }
19 }
```

- switch(*kifejezés*) *utasítás*
- *kifejezés* egész típusú
- *utasítás* tartalmazhat
  - több case *konstans-kifejezés: utasítás*-t,
  - nulla vagy egy default: *utasítás*-t
- végrehajtás leáll:
  - switch blokkjának végén
  - az első break utasításnál
- *konstans-kifejezés* egész típusú
- *kifejezés* és *konstans-kifejezés* értékeinek összehasonlítása
- több case címke is címkézheti ugyanazt az utasítást, de minden címkének egyedinek kell lennie
- switch utasítások egymásba ágyazhatóak