Programozási tételek és C++

Alábbiakban néhány kódolást könnyítő lehetőségét nézzük meg a a C++ nyelvnek. Elsősorban a programozási tételek kódolására gondolunk. A „hagyományos" kódolással kezdjük, amely a továbbiak alapja lesz. Ez szolgál kiinduló pontként a további kódolásrövidítő módszereknek: makróra építőnek és a tételeket függvényként implementálónak. Példaként a (lineáris) keresés tételt nézzük.

Vázlat

Specifikáció$_1$ – a szokásos részletezésű specifikáció
Specifikáció$_2$ – a függvénnyel tömörített utófeltételű specifikáció
Algoritmus
Kód$_1$ – „normál" kódolás
Kód$_2$ – makrós kódolás
Kód$_3$ – függvényes kódolás
Tételalkalmazások

Kifejtés

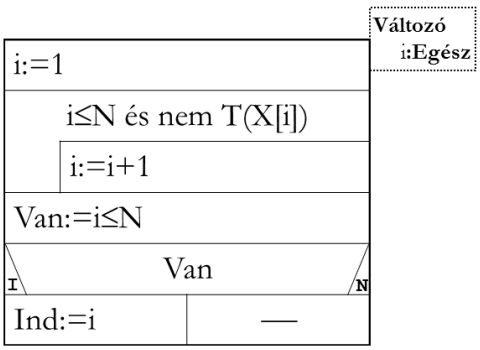
Specifikáció$_1$ – Keresés

- **Bemenet**: N:Egész,
  X:Tömb[1..N:Valami]
- **Kimenet**: Van:Logikai, Ind:Egész
- **Előfeltétel**: $N \geq 0$
- **Utófeltétel**: $Van = \exists i\ (1 \leq i \leq N)$: T(X[i]) és
  $Van \rightarrow 1 \leq Ind \leq N$ és T(X[Ind])

Specifikáció$_2$ – Keresés

- **Utófeltétel**: (Van,Ind)=VanÉsInd(1..N,X[·],T(·))
- **Definíció**: VanÉsInd:Egész^2 ×Valami*×(Valami→Logikai)→Logikai×Egész
  VanÉsInd(e..v,x[·],t(·)):=(van,ind): van=∃i(e≤i≤v): t(x[i]) és
  van→e≤ind≤v és t(x[ind])

Algoritmus – Keresés



Kód$_1$ – „normál” kódolás – Keresés

A programparamétereknek –a specifikáció bemeneti és kimeneti részei alapján– az alábbi deklarációkat feleltetjük meg:

```
//Bemeneti paraméterek:
int N;
const int maxN=100;
valami X[maxN];
//Kimeneti paraméterek:
bool Van;
int Ind;
//Tulajdonságfüggvény prototípusa:
bool T(valami x);
```

A lényegi rész kódját az algoritmusból származtatjuk, a „természetes” kódolási szabályokkal:

```
//a „lokális” ciklusváltozó a tárolt 0-indexű elemnél kezdi
int i=0;
while (i<N && !T(X[i]))
{
  ++i;
}
Van=i<N;
if (Van)
{
  Ind=i;
}
```

Kód$_2$ – makrós kódolás – Keresés

Alapgondolat: minden tételnek megfeleltetünk egy makrót, amely kifejtését –mint mechanikusan alkalmazandó átírási szabályt– rábízzuk a preprocesszorra. E módszernek talán az a legfontosabb mondani valója, hogy a programozási folyamatban a tételek kóddá alakítása valóban mechanikusan elvégezhető feladat.

A **makródefiníció** egyetlen sorból áll, és az alábbi szintaxisú:

#define `makrónév [ paraméterlista ] törzs`

ahol

`makrónév` – *egy alfanumerikus jelsorozat*
`paraméterlista` – *kerek zárójelek közé írt, vesszővel elválasztott* `paraméter`*-sorozat*
`paraméter` – *egy definición belül unikális alfanumerikus jelsorozat*
`törzs` – *jelsorozat, amelyben felbukkannak a* `paraméter`*-ek*

A […] között rész elmaradhat. A mi céljaink eléréséhez mindig kellenek paraméterek.

A **makróhívás** egyszerűen a `makrónév` kiegészítve a konkrétan behelyettesítendő `paraméterlista`-val, természetesen ha van egyáltalán paramétere.

Íme, a keresés tétel Kód$_1$-beli kódolás mintáját követő változata. Nincs sortörés, most csak nyomdatechnikai okok miatt, és a könnyebb olvashatóság érdekében tagoltuk sorokra:

```
#define VanEsInd(cv,e,v,felt,van,ind)
   cv=e;
   while(cv<=v && !(felt)) {++cv;}
   van=cv<=v;
   if(van) {ind=cv;}
```

1Def

Megjegyzések a fenti definícióhoz:

1. A makró definiálásánál a **Specifikáció$_2$**-ban előforduló függvény szolgál ötletadóul.
2. Látható, hogy a paraméterek között az algoritmus **kimeneti változói is** felbukkannak; azaz a kiindulópontul szolgáló függvénytől eltérően nem függvényként definiáljuk a makrót, ami persze nem is meglepő, hiszen a majdani hívásnál nem lesz módunk függvényként alkalmazni. Valóban mechanikus kódbehelyettesítésként fog működni.
3. A `felt` paraméter a **sorozat és a tulajdonság** szerepét egyszerre tölti be; ez egy formális szimbólum, amely lehetővé teszi, hogy helyére tetszőleges logikai formulát (akár tulajdonságfüggvényt aktuális paraméterestül) illeszthessünk.
4. A látszólag fölösleges **ciklusváltozó** azért kell, mert a bemeneti paramétereket jelentő konkrét formulákban (pl. tömbelemben) ennek szerepelni kell.

Egy hívási példával világítjuk meg a fentieket:

```
VanEsInd(i,0,N-1,T(X[i]),Van,Ind)
```
1Hiv

Ezt az alábbi kóddal helyettesíti a preprocesszor (persze egyetlen sorban):

```
i=0;
while(i<=N-1 && !(T(X[i]))) {++i;}
Van=i<=N-1;
if(Van) {Ind=i;}
```
1Kif

Ezt összevetve a **Kód$_1$**-ben szereplő kóddal, elégedettek lehetünk.

Ha közelíteni akarjuk a specifikációbeli definícióhoz, akkor az alábbi módon is definiálhatjuk a makrót (piros a különbség):

```
#define VanEsInd(cv,e,v,x,t,van,ind)
   cv=e;
   while(cv<=v && !t(x[cv])) {++cv;}
   van=cv<=v;
   if(van) {ind=cv;}
```
2Def

Ennek hívása az alábbi lenne:

```
VanEsInd(i,0,N-1,X,T,Van,Ind)
```
2Hiv

Hogy ugyanazt a kódot kapnánk a kifejtés után, az nyilvánvaló.
Az alkalmazás rugalmasságát tekintve azonban az első megoldás előnyösebb. Ugyanis ez megengedi az alábbi –közvetlen kifejezést tartalmazó– hívást is, amíg a második –értelemszerűen– nem:

```
VanEsInd(i,0,N-1,X[i]%2==0,Van,Ind)
```
1Hiv2

Ez viszont nem működik:

```
VanEsInd(i,0,N-1,X,X[i]%2==0,Van,Ind)
```
2Hiv2

Ugyanis a mechanikus behelyettesítés során az alábbi C++ nyelvbeli non sence kódot kapnánk:

```
i=0;
while(i<=N-1 && !(X[i]%2==0(X[i])) {++i;}
Van=i<=N-1;
if(Van) {Ind=i;}
```
2Kif2

Második példaként a keresés index helyett **érteket visszaadó változat**át definiáljuk: az eredményt az `Ert` változó tartalmazza. Ebben a változatban a sorozat elemei nem csak a

tulajdonság függvényben bukkannak föl, ezért egy őket megtestesítő paramétert is szerepeltetni kell:

```
#define VanEsErt(cv,e,v,sorelem,felt,van,ert)
   cv=e;
   while(cv<=v && !(felt)) {++cv;}
   van=cv<=v;
   if(van) {ert=sorelem;}
```
3Def

Egy hívási példa, amely az első valódi osztóját keresi az N-nek:

```
int oszto;
bool oszthatoE;
VanEsErt(i,2,sqrt(N),i,N%i==0,oszthatoE,oszto)
```
3Hiv

Egy másik hívási példa, amely egy olyan értékét keresi meg a tömbnek, amelyet követő tömbelem 2-vel nagyobb nála:

```
int Szam[10];
bool Van;
int Ert;
… //a Szam tömb feltöltése
VanEsErt(i,1,9,Szam[i],Szam[i+1]-Szam[i+1]==2,Van,Ert)
```
3Hiv

Kód3 – függvényes kódolás – Keresés

A függvény-alapú megoldásnál a tételeket valódi C++ függvényként kell implementálni. Itt a nehézséget maga a nyelv okozza, nem a preprocesszor képességei. Két problémára kell gyógyírt találni:

1. Az olyan tételeknél, amelyek valamilyen függvényre építenek (elsősorban a tulajdonság függvényre gondolunk), elegáns lenne, ha a tételfüggvény a kellő **függvényt paraméterként** venné át.
2. A konkrét feladatokban az érintett tételeket más és más típusú adatszerkezetekre kell alkalmazni. Ez azt jelenti, hogy ugyanaz a tételparaméter más és más típusúként kellene deklarálni, vagyis a **típussal való paraméterezést** is meg kell oldani.

Megoldás:

1. Korlátozzuk vizsgálatunkat csupán a tulajdonság függvényekre! Az alapötlet az, hogy a függvényparaméter a **függvény kezdőcímének átvételét** jelentse. Természetesen megállapodunk abban, hogy a kezdőcím valamely konkrét szignatúrájú függvényekhez tartozik.

   ```
   bool (*bool2tip)(tip x)
   ```

   ez a `bool2tip` nevű függvény(paraméter) **cím**ét jelenti, ahol a `tip` helyére valamely konkrét típus képzelendő. Alább éppen azt fogjuk leírni, hogy miként kerülhet ide –kódismétlés nélkül– tetszőleges típus.
2. A típussal paraméterezést **sablondefiniálás**sal oldhatjuk meg:

   ```
   template<class tip> fejsor
   ```

   Itt a **`template`** kulcs-szó jelzi a sablondefiníció kezdetét. A **`<class tip>`** arra utal, hogy egy típus (pontosabban egy osztály, ami típus általánosítása) lesz a sablon paramétere, amelyre a `tip` formális névvel fogunk hivatkozni. `A fejsor` helyére egy függvény szokásos fejsora írandó, amelyben azonban –és éppen ez a különlegesség– szerepelhet valamely típus helyett a `tip` formális típusparaméter.

Példa:

```
template<class tip> tip Szumma(int tol, int ig, const tip t[]);
```

és egy másik:

```
template<class tip> bool VanE(int tol, int ig, const tip t[],
                              bool (*tulFv)(tip e));
```

Az első példa az összegzés tétel általánosított függvényének prototípusa. 3. paramétere egy `tip` típusú elemekből álló tömb, amelynek `t[i] i=tol..ig` elemeire vonatkozik az „általánosított összegzés". A számítás eredménye is `tip` típusú.

A függvény definíciója az alábbi:

```
template<class T> T Szumma(int tol, int ig, const T t[])
{
  T s=0;
  for (int i=tol;i<=ig;++i)
  {
    s+=t[i];
  }
  return s;
}
```

A második példa az eldöntés tétel egyik alakjának a prototípusa. Magyarázatra csupán az utolsó paramétere szorul: ez a tulajdonság függvény címét jelentő formális paraméter, amely függvény szignatúrája az alábbi:

`tulFv: tip→bool`

Ilyen függvények lehetnek az alábbiak:

```
bool paros(int n)
{
  return n%2==0;
}
```

```
bool nemUresSzoveg(string s)
{
  return s>"";
}
```

A függvény definíciója az alábbi:

```
template<class T>  bool VanE(int tol, int ig, const T t[],
                             bool (*tulFv)(T e))
{
   int i=tol;
   while (i<=ig && !tulFv(t[i]))
   {
     ++i;
   }
   return i<=ig;
}
```

Tételalkalmazások

Egy komplett programmal mutatjuk be a makró-alapú tételalkalmazásokat. Vannak sorok, amelyek hosszúk miatt eltörtek. Hogy ez ne okozzon gondot, a folytatás-sorokat egy sor előtti vonallal megjelöltük.

```
//Szlávi Péter
//SZPKAFT.ELTE
//szlavip@elte.hu
//FELADAT:
//  Algoritmusrövidítő makrók a programozásban.

#include <iostream>
#include <stdlib.h>//system-hez
```

```
#include <math.h>//sqrt-hez

using namespace std;

//algoritmus-minták makrókkal (ahol a parameter képlet is lehet,
//ott célszerű zárójelezetten beilleszteni):
#define ForCiklus(cv,tol,ig) for(int cv=(tol);cv<=(ig);++cv) /*ciklusmag nélkül!!!*/
#define SorozatKi(cv,tol,ig,sorozatElem) for(int cv=(tol);cv<=(ig);++cv)
{cout<<cv<<".:"<<sorozatElem<<endl;}
#define Hany(cv,tol,ig,felt,db) db=0; ForCiklus(cv,tol,ig) {if(felt) ++db;}
#define Szumma(cv,tol,ig,sorozatElem,osszeg) osszeg=0; ForCiklus(cv,tol,ig)
 {osszeg+=sorozatElem;}
#define VanE(cv,tol,ig,felt,van) cv=(tol); while(cv<=(ig) && !(felt)) {++cv;}
van=cv<=(ig);
#define MelyInd(ind,tol,felt) ind=(tol); while(!(felt)) {++ind;}
#define VanEsInd(cv,tol,ig,felt,van,ind) cv=tol; while(cv<=ig && !(felt)) {++cv;}
van=cv<=ig; if(van) {ind=cv;}
#define VanEsErt(cv,e,v,sorElem,felt,van,ert) cv=e; while(cv<=v && !(felt)) {++cv;}
van=cv<=v; if(van) {ert=sorElem;}
#define MaxInd(cv,tol,ig,tomb,mxI) mxI=tol; ForCiklus(cv,tol+1,ig)
{if(tomb[cv]>tomb[mxI]) mxI=cv;}
#define MinInd(cv,tol,ig,tomb,mnI) mnI=tol; ForCiklus(cv,tol+1,ig)
{if(tomb[cv]<tomb[mnI]) mnI=cv;}
#define MnxInd(cv,tol,ig,tomb,r,mI) mI=tol; ForCiklus(cv,tol+1,ig) {if(tomb[cv] r
tomb[mI]) mI=cv;}
#define BillreVar cout << endl; system("pause");
#define UjLap system("cls");

int main()
{
  cout << "'FORCIKLUS' makro: ----------------------------------------------" << endl;
  cout << "ForCiklus(i,1,5) {cout<<i<<endl;}:" << endl;
  ForCiklus(i,1,5)
  {
    cout << i << endl;
  }
  BillreVar

  UjLap
  cout << "'SorozatKi' makro: ----------------------------------------------" << endl;
  const int HoHossz[12]={31,28,31,30,31,30,31,31,30,31,30,31};
  cout << "HoHossz[0..11]:" << endl;
  SorozatKi(i,0,11,HoHossz[i])
  cout << "HoHossz[1..12]:" << endl;
  SorozatKi(i,1,12,HoHossz[i-1])
  BillreVar

  UjLap
  cout << "'Hany' makro: ---------------------------------------------------" << endl;
  int db;
  Hany(i,0,11,HoHossz[i]>30,db)
  cout << "Hany HoHossz[0..11]>30? Darab=" << db << endl;
  Hany(i,0,11,HoHossz[i]==30,db)
  cout << "Hany HoHossz[0..11]=30? Darab=" << db << endl;
  Hany(i,0,11,HoHossz[i]<30,db)
  cout << "Hany HoHossz[0..11]<30? Darab=" << db << endl;
  BillreVar

  UjLap
  cout << "'Szumma' makro: -------------------------------------------------" << endl;
  int ossz;
  Szumma(i,1,5,i,ossz)
  cout << "Szumma(1..5)=" << ossz << endl;

  Szumma(i,1,12,HoHossz[i-1],ossz)
  cout << "Szumma(HoHossz[1..12])=" << ossz << endl;
  BillreVar

  UjLap
  cout << "'VanE' makro: ---------------------------------------------------" << endl;
  ForCiklus(N,2,7)
  {
    int i;
    bool oszthatoE;
    VanE(i,2,N/2,N%i==0,oszthatoE)
    cout << N << " prim-e: ";
    if(oszthatoE)
    {
```

```
      cout << "nem ";
    }
    cout << "prim." << endl;
  }

  cout << "'MelyInd' makro: ----------------------------------------" << endl;
  ForCiklus(N,2,7)
  {
    int oszto;
    MelyInd(oszto,2,N%oszto==0)
    cout << "LKO(" << N << ")= " << oszto << endl;
  }

  cout << "'VanEsInd' makro: ----------------------------------------" << endl;
  ForCiklus(N,17,25)
  {
    int i,oszto;
    bool oszthatoE;
    VanEsInd(i,2,sqrt(N),N%i==0,oszthatoE,oszto)
    cout << N << " oszthato-e: ";
    if(oszthatoE)
    {
      cout << "igen; LKO(" << N << ")= " << oszto << "." << endl;
    }
      else
    {
      cout << "nem." << endl;
    }
  }

  cout << "'VanEsErt' makro: ----------------------------------------" << endl;
  ForCiklus(N,1,4)
  {
    int i,hanyNapos;
    bool vanE;
    VanEsErt(i,1,11,HoHossz[i],HoHossz[i+1]-HoHossz[i]==N,vanE,hanyNapos)
    cout << "Van-e olyan ho, amelyet kovetonek " << N << " nappal tobb napja van:";
    if(vanE)
    {
      cout << "igen; megpedig a " << hanyNapos << "." << endl;
    }
      else
    {
      cout << "nincs." << endl;
    }
  }
  BillreVar

  UjLap
  cout << "'MaxInd/MinInd' makro: ----------------------------------------" << endl;
  int maxI;
  MaxInd(i,0,11,HoHossz,maxI)
  cout << "MaxInd(HoHossz[1..12])=" << maxI+1 << ", ";
  cout << HoHossz[maxI] << " napos." << endl;

  int minI;
  MinInd(i,0,11,HoHossz,minI)
  cout << "MinInd(HoHossz[1..12])=" << minI+1 << ", ";
  cout << HoHossz[minI] << " napos." << endl;

  MnxInd(i,0,11,HoHossz,<,minI)
  cout << "MnxInd(HoHossz[1..12],<)=" << minI+1 << ", ";
  cout << HoHossz[minI] << " napos." << endl;
  BillreVar

  return 0;
}
```

Egy komplett programmal mutatjuk be a függvény-alapú tételalkalmazásokat.

```
//Szlávi Péter
//SZPKAFT.ELTE
//szlavip@elte.hu
//FELADAT:
//  Függvények, függvénytípus, függvény mint paraméter.
```

```cpp
#include <iostream>
#include <stdlib.h>//az exit-hez, a 10.05-höz

using namespace std;

//Tulajdonság függvények: ----------------------------------------------------------------
---
bool parosE(int n);                          //paros:int->bool konkrét függvény
template<class T> bool _10nelNagyobbE(T n);//_10nelNagyobbE:T->bool sablon függvény
bool nagyKezdetuE(string s);                 //nagyKezdetuE:string->bool konkrét függvény
//beolvasás:
template<class T> void be(string kerd, T &e, bool (*tulFv)(T e),string hibaUzenet);
//tételfüggvények:
template<class T> T Szumma(int tol, int ig, const T t[]);
template<class T> bool VanE(int tol, int ig, const T t[], bool (*tulFv)(T e));

int main()
{
//beolvasás:
  int m;
  be("Kerek egy paros egeszet:", m, parosE, "Ez nem paros egesz! Kerem ujra!");
  be("Kerek egy 10-nel nagyobb egeszet:", m, _10nelNagyobbE,
     "Ez nem 10-nel nagyobb egesz! Kerem ujra!");
  double x;
  be("Kerek egy 10-nel nagyobb valosat:", m, _10nelNagyobbE,
     "Ez nem 10-nel nagyobb valos! Kerem ujra!");
  string szo;//az inputnak csak az elsö szavát tudjuk beolvasni alább!
  be("Kerek egy szot nagybetuvel kezdve:", szo, nagyKezdetuE,
     "Ez nem nagybetuvel kezdodo szo! Kerem ujra!");

//tételfüggvények:
  const int egeszek[]={1,3,5,7,9,8,6,4,2};
  const int N=(sizeof egeszek)/sizeof(int);
  cout << "Az egeszek osszege:" << Szumma(0,N-1,egeszek) << endl;

  cout << "Az egeszek kozott ";
  if (!VanE(0,N-1,egeszek,parosE))
  {
    cout << "nem ";
  }
  cout << "talalhato paros." << endl;

  cout << "Az egeszek kozott ";
  if (!VanE(0,N-1,egeszek,_10nelNagyobbE))
  {
    cout << "nem ";
  }
  cout << "talalhato 10-nel nagyobb." << endl;

  return 0;
}

bool parosE(int n)//paros:int->bool konkrét függvény
{
  return (n%2==0);
}

template<class T> bool _10nelNagyobbE(T n)//_10nelNagyobbE:T->bool konkrét függvény
{
  return n>10;
}

bool nagyKezdetuE(string s)//nagyKezdetuE:string->bool konkrét függvény
{
  return s.length()>0 && isupper(s.at(0));
}

template<class T> T Szumma(int tol, int ig, const T t[])
{
  T s=0;
  for (int i=tol;i<=ig;++i)
  {
    s+=t[i];
  }
  return s;
}
```

```
template<class T> bool VanE(int tol, int ig, const T t[], bool (*tulFv)(T e))
{
   int i=tol;
   while (i<=ig && !tulFv(t[i]))
   {
     ++i;
   }
   return i<=ig;
}

template<class T> void be(string kerd, T &e, bool (*tulFv)(T e),string hibaUzenet)
{
   bool hiba;
   string tmp;
   do{
     cout << kerd; cin >> e;
     hiba=cin.fail() || !tulFv(e);
     if(hiba)
     {
       cout << hibaUzenet << endl;
       cin.clear(); getline(cin,tmp,'');
     }
   }while(hiba);
}
```