C++ függelék

Juhász Tibor–Tóth Bertalan: Programozási ismeretek kezdő versenyzőknek
(Műszaki Könyvkiadó, 2017)
Kiegészítés
Összeállította: Tóth Bertalan

Tartalom

1. Sztring feldarabolása elválasztó karakter mentén

   A szükséges fejállományok: <iostream>, <string>, <sstream>, <vector>

```
vector<string> Split(const string &s, char elvalaszto) {
    vector<string> elemek;
    stringstream ss(s);
    string elem;
    while (getline(ss, elem, elvalaszto)) {
        elemek.push_back(elem);
    }
    return elemek;
}
```

2. Olyan szövegfájl sorainak felolvasása, amelyekben előre nem ismert darabszámú adat található

   A szükséges fejállományok: <iostream>, <fstream>

```
int adat;
ifstream fajlbe("Adatok.txt");
while (fajlbe >> adat) {
   cout << adat << " ";
   if (fajlbe.get()=='\n')  // elértük a sor végét?
      cout << endl;
}
fajlbe.close();
```

3. Két tömb (a és b) együttes rendezése az első tömb (a) adatai alapján

   A szükséges fejállományok: <iostream>, <algorithm>

```
int main() {
    const int n=7;
    int a[n] = { 1, 4, 3, 0, 2, 12, 23};
    int b[n] = { 10, 40, 30, 0, 20, 120, 230};
    for (int i=0; i<n-1; i++) {
        for (int j=i+1; j<n; j++) {
            if (a[i]>a[j]) {
               swap(a[i], a[j]);
               swap(b[i], b[j]);
            }
        }
    }
    for (int i=0; i<n; i++) {
        cout<<a[i]<< " " << b[i] << endl;
    }
    cout << endl;
}
```

4. Szövegsor darabolása a *Split()* függvény kétszeri hívásával

A szükséges fejállományok: <iostream>, < vector>, <cstdlib>

```
int main() {
   int csapat1, csapat2, gol1, gol2;
   vector<string> temp, golok;

   string sor="1 3 2:4";
   temp = Split(sor,' ');
   csapat1 = atoi(temp[0].c_str());
   csapat2 = atoi(temp[1].c_str());

   golok = Split(temp[2],':');
   gol1 = atoi(golok[0].c_str());
   gol2 = atoi(golok[1].c_str());
}
```

5. Egydimenziós tömbök a C++ programokban

A szükséges fejállományok: <iostream>, < vector>

```
int main() {
    // Statikus egydimenziós tömb kezdőérték-adással
    // A tömb csak a main() függvényből való kilépéskor törlődik
    const int meret = 7;
    int a[meret] = {1, 2, 3, 5, 8, 13};
    a[meret-1] = a[meret-2]+a[meret-3];

    // Dinamikus egydimenziós tömb, mutató felhasználásával
    int *p = new int[meret] {1, 2, 3, 5, 8, 13};
    p[meret-1] = p[meret-2]+p[meret-3];
    // A dinamikus tömb törlése
    delete[] p;

    // A vector sablon felhasználásával az elemek a vektor törlése nélkül is
    // eltávolíthatók, maga a vektor azonban nem törölhető
    vector<int> v {1, 2, 3, 5, 8, 13}; // 6-elemű vektor
    v.resize(v.size()+1);              // újraméretezzük
    v[v.size()-1] = v[v.size()-2] + v[v.size()-3];
    v.clear();                         // a vektornak nincs egyetlen eleme sem
}
```

6. Sztring – egész szám átalakítások

A szükséges fejállományok: <iostream>, <string>, <cstdlib>

```
string IntToStr(unsigned long long szam, unsigned char alap) {
   static char szamjegyek[36+1] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ" ;
   string szamb = "";
   int szamjegy;
   while (szam > 0) {
      szamjegy = szam % alap;
      szamb = szamjegyek[szamjegy] + szamb;
      szam /= alap;
   }
   return szamb == "" ? "0" : szamb;
}
```

```
int main () {
  string str10 = "2014 C++ Szabvany";
  int adat;

  adat = atoi(str10.c_str());
  cout << adat << endl; // 2014
  adat = stoi(str10);
  cout << adat << endl; // 2014

  string str20 = IntToStr(38432, 20) + "0";
  cout << str20 << endl; // 4G1C00

  string str16 = "131CD2A";
  string str02 = "-10011000111";

  size_t p;
  int egesz10 = stoi (str10, &p);
  cout << egesz10 << " : " << str10.substr(p) << endl; // 2014 :  C++ Szabvany
  int egesz02 = stoi (str02, nullptr, 2);
  unsigned long egesz20 =  stoul (str20, nullptr, 20);
  unsigned long long egesz16 = stoll (str16, nullptr, 16);

  string szamok = to_string(egesz10) + ", " + to_string(egesz16) + ", ";
  szamok += to_string(egesz02) + ", " +to_string(egesz20);
  cout << szamok << endl; // 2014, 20041002, -1223, 768640
  cin.get();
}
```

7. C++11 véletlenszám-generálási példa

A szükséges fejállományok: <iostream>, <random>, <ctime>

```
int main() {
    random_device rd;  // véletlen eszköz
    mt19937 gen(rd()); // a választott generátor
    gen.seed(time(0)); // véletlen kezdőérték
    // egyenletes eloszlás a [0, 1) intervallumban
    uniform_real_distribution<double> nd(0, 1);
    for (int i=0; i<10; i++)
        cout << nd(gen) << endl; // véletlen szám létrehozása
}
```

8. Hagyományos dátum-/időkezelés a C++-ban

A szükséges fejállományok: <iostream>, < clocale>, <ctime>

```
// Várakozik ms ezredmásodpercig (Windows alatt)
void Varakozik(clock_t ms ) {
   clock_t cel;
   cel = ms + clock();
   while( cel > clock() );
}

int main (void) {
  setlocale(LC_ALL, "Hun");
  const int pmeret = 123;
  char puffer[pmeret];
  time_t most;
  tm *helyiido;
```

```
    // az aktuális idő lekérdezése
    most = time (NULL);

    // átalakítása helyi idővé
    helyiido = localtime (&most);

    // kiírás szabványos formátumban
    cout << asctime (helyiido) << endl;

    // kiírás saját formátumban
    strftime (puffer, pmeret, "Ma %A, %B %d. van.\n", helyiido);
    cout << puffer << endl;
    strftime (puffer, pmeret, "Az idő: %I:%M:%S %p.\n", helyiido);
    cout << puffer << endl;

    // ---------------------------------------------------------------------
    // Milyen napra esik az adott dátum?
    int ev, ho, nap;
    tm datum  = {0};
    time_t akkor;
    cout << "év :  "; cin >> ev;
    cout << "hó :  "; cin >> ho;
    cout << "nap:  "; cin >> nap; cin.get();

    datum.tm_year = ev - 1900;
    datum.tm_mon = ho - 1;
    datum.tm_mday = nap;
    akkor = mktime(&datum);
    if (akkor == -1)
        cout << "hibás dátum" << endl;
    else {
        strftime(puffer, pmeret, "%A", &datum);
        cout << "az év " << datum.tm_yday  <<". napja, " << endl;
        cout << "a hét " << datum.tm_wday  <<". napja, " << endl;
        cout << puffer << endl;
        // A két időpont közötti másodpercek száma:
        cout << "Az eltelt másodpercek: " << fixed << difftime(akkor, most);
        cout << endl;
    }

    // ---------------------------------------------------------------------
    // futásidő mérése:
    cout << endl;
    clock_t kezdete = clock();
    Varakozik(1223);
    clock_t vege = clock();
    clock_t orautesek = vege - kezdete;
    double masodpercek = orautesek / (double) CLOCKS_PER_SEC;
    cout << "futásidő: " << masodpercek << endl;

    cin.get();
    return 0;
}
```

9. C/C++11 vegyes dátum- és időkezelés

A szükséges fejállományok: <iostream>, < clocale>, <ctime>, <chrono>

```
// Várakozik ms ezredmásodpercig
void Varakozik(unsigned long ms ) {
   typedef chrono::high_resolution_clock ora;
   ora::time_point kezdete = ora::now();
   ora::time_point vege = kezdete;
   while (chrono::duration_cast<chrono::milliseconds>
                                (vege - kezdete).count() < ms) {
      vege = ora::now();
   }
}

typedef chrono::system_clock ora;

int main (void) {
  setlocale(LC_ALL, "Hun");
  const int pmeret = 123;
  char puffer[pmeret];
  ora::time_point most;
  time_t most_t;
  tm *helyiido;

  // az aktuális idő lekérdezése
  most = ora::now();
  most_t = ora::to_time_t(most);
  // átalakítása helyi idővé
  helyiido = localtime (&most_t);

  // kiírás szabványos formátumban
  cout << asctime (helyiido) << endl;

  // kiírás saját formátumban
  strftime (puffer, pmeret, "Ma %A, %B %d. van.\n", helyiido);
  cout << puffer << endl;
  strftime (puffer, pmeret, "Az idő: %I:%M:%S %p.\n", helyiido);
  cout << puffer << endl;

  // ---------------------------------------------------------------------
  // Milyen napra esik az adott dátum?
  int ev, ho, nap;
  tm datum  = {0};
  ora::time_point akkor;
  time_t akkor_t;
  cout << "év :  "; cin >> ev;
  cout << "hó :  "; cin >> ho;
  cout << "nap:  "; cin >> nap; cin.get();

  datum.tm_year = ev - 1900;
  datum.tm_mon = ho - 1;
  datum.tm_mday = nap;
  akkor_t = mktime(&datum);
  if (akkor_t == -1)
      cout << "hibás dátum" << endl;
  else {
      strftime(puffer, pmeret, "%A", &datum);
      cout << "az év " << datum.tm_yday  <<". napja, " << endl;
      cout << "a hét " << datum.tm_wday  <<". napja, " << endl;
      cout << puffer << endl;
```

```
    // A két időpont közötti másodpercek száma:
    akkor = ora::from_time_t(akkor_t);
    chrono::duration<double> masodpercek = akkor-most;
    cout << "Az eltelt másodpercek: " << fixed;
    cout << masodpercek.count();
    cout << endl;
  }
  // ---------------------------------------------------------------------------
  // futásidő mérése:
  cout << endl;
  ora::time_point kezdete = ora::now();
  Varakozik(1223);
  ora::time_point vege= ora::now();
  unsigned long ms = chrono::duration_cast<chrono::milliseconds>
                     (vege - kezdete).count();
  cout << "futásidő: " << ms/1000.0 << endl;
  cin.get();
  return 0;
}
```

10. Magyar szövegek rendezése – 1.
    A C++ alapértelmezés szerint helyi beállapításaival és saját kódtáblával

A szükséges fejállományok: <iostream>, < clocale>, <string>, <algorithm>

```
static string sorrend_Abc = "aAáÁbBcCdDeEéÉfFgGhHiIíÍjJkKlLmMnNoO"
                            "óÓöÖőŐpPqQrRsStTuUúÚüÜűŰvVwWxXyYzZ";

bool Azonos(char cha, char chb) {
  int pa = sorrend_Abc.find(cha);
  int pb = sorrend_Abc.find(chb);
  return (pa/2) == (pb/2);
}

bool Kisebb(string a, string  b) {
  bool eredmeny = false;
  int i = 0;
  while ( (a[i]!=0) && (b[i]!=0) && Azonos(a[i],b[i]) ) {
    i++;
  }
  int pa = sorrend_Abc.find(a[i]);
  int pb = sorrend_Abc.find(b[i]);
  if ((pa==-1) || ((pa!=-1 && pb!=-1) && ((pa/2) < (pb/2))) )
     eredmeny = true;
  return eredmeny;
}

int main() {
  setlocale(LC_ALL, ".1250");
  string nevek[] =  {"Adél", "abigél", "Ádám", "ábel", "Ágnes", "bandi", "Adél",
                     "Bálint", "Aladár","Benő"};
  sort(begin(nevek), end(nevek),  Kisebb);
  for (const string& s : nevek)
    cout << s << endl;
  cout << endl;
  cin.get();
  return 0;

}
```

```
abigél
Adél
Adél
Aladár
ábel
Ádám
Ágnes
bandi
Bálint
Benő
```

11. Magyar szövegek rendezése – 2.
A C++ magyar („hu-HU”) helyi beállapításainak alkalmazásával (VC++ 2013-ban)

A szükséges fejállományok: <iostream>, < clocale>, <string>, <locale>, <algorithm>

```
bool Kisebb(string a, string b) {
  locale loc("hu-HU");
  const collate<char>& coll = use_facet<collate<char> >(loc);
  int result = coll.compare(a.data(), a.data() + a.length(),
                            b.data(), b.data() + b.length());
  return result < 0;
}

int main() {
  setlocale(LC_ALL, "hu-HU");
  string nevek[] = {"Adél", "abigél", "Ádám", "ábel", "Ágnes", "bandi",  "Adél",
                    "Bálint", "Aladár","Benő"};
  sort(begin(nevek), end(nevek), Kisebb);
  for (const string& s : nevek)
    cout << s << endl;
  cin.get();
  return 0;
}
```

```
ábel
abigél
Ádám
Adél
Adél
Ágnes
Aladár
Bálint
bandi
Benő
```

12. A Visual Basic *like* műveletének megfelelő C++11 program
az STL reguláris kifejezések könyvtárának a használatával

A szükséges fejállományok: <iostream>, < regex>, <string>

```
bool Like(const string& str, const regex& minta) {
  return regex_match(str,minta);
}

int main() {
  regex minta(".[0-9][a-ft-z].*[^aez]");
  if (Like("x3zbbbbbbf",minta))  // hasonlít
    cout<<"hasonlít"<<endl;
  else
    cout<<"nem hasonlít"<<endl;

  if (Like("x3z",minta))        // nem hasonlít
    cout<<"hasonlít"<<endl;
  else
    cout<<"nem hasonlít"<<endl;
}
```

13. Rendezett halmazfelsorolást előállító program időméréssel
A megoldás a sort() és az unique() algoritmusokra épül.

A szükséges fejállományok: <iostream>, < ctime>, <cstdlib>, <vector>, <algorithms, <chrono>

```
class Stopper {
private:
  chrono::time_point<chrono::high_resolution_clock> kezdete, vege;
public:
  void Start() { kezdete = vege = chrono::high_resolution_clock::now(); }
  void Stop() { vege = chrono::high_resolution_clock::now(); }
  double ElteltIdo() {
    auto eltelt = chrono::duration_cast<chrono::milliseconds>(vege-kezdete);
    return eltelt.count();
  }
};
```

```
int main() {
  unsigned int m = 10000000;
  vector<int>tomb(m, 0);
  srand(time(0));
  for (int& elem: tomb)
    elem = rand();
  Stopper stopper;

  stopper.Start();
  sort(begin(tomb), end(tomb));
  auto utolso = std::unique(begin(tomb), end(tomb));
  tomb.erase(utolso, end(tomb));
  stopper.Stop();

  cout << "Eltelt ido" << stopper.ElteltIdo() << " ms\n";
  cout << "A megmaradt elemk szama:" << tomb.size() << endl;
  cin.get();
  return 0;
}
```

14. Szöveges állomány teljes tartalmának sztringbe olvasása

A szükséges fejállományok: <iostream>, < string>, < fstream>, < sstream>

```
int main() {
  ifstream fajlbe("Szövegfájl.txt");
  stringstream puffer;
  puffer << fajlbe.rdbuf();
  fajlbe.close();
  string szoveg = puffer.str();
  cout << szoveg << endl;
  cin.get();
  return 0;
}
```

15. Struktúra változók biztonságos azonosság-vizsgálata C++-ban

A szükséges fejállományok: <iostream>, < string>

```
struct TDiak {
  string nev;
  bool fiu;
};

bool Megegyeznek(const TDiak& d1, const TDiak& d2) {
  return (d1.nev == d2.nev) && (d1.fiu == d2.fiu);
}

bool operator==(const TDiak& d1, const TDiak& d2) {
  return (d1.nev == d2.nev) && (d1.fiu == d2.fiu);
}
```

```
int main() {
  TDiak diak1 = {"Laci", true};
  TDiak diak2;
  diak2.nev = "Laci";
  diak2.fiu = "true";
  cout << Megegyeznek(diak1, diak2) << endl; // 1 -> igen
  cout << (diak1==diak2) << endl;            // 1 -> igen
  cin.get();
  return 0;
}
```

16. Egydimenziós tömbök átadása függvényeknek

A szükséges fejállományok: <iostream>, < vector>

```
void Feltolt(int tomb[], int meret, int adat) {
  for (int i=0; i<meret; i++)
    tomb[i] = adat;
}

void Feltolt(vector<int>& tomb, int adat) {
  for (int& elem : tomb)
    elem = adat;
}

vector<int> Letrehoz_es_Feltolt(int meret, int adat) {
  vector<int> t(meret, adat);
  return t;
}

int main() {
  const int elemszam = 12, tesztadat = 123;
  // Statikus tömb
  int stomb[elemszam];
  Feltolt(stomb, elemszam, tesztadat);
  for (int elem : stomb)
    cout << elem << " "; cout << endl;

  // Dinamikus tömb
  int *dtomb = new int[elemszam];
  Feltolt(dtomb, elemszam, tesztadat);
  for (int i=0; i<elemszam; i++)
    cout << dtomb[i] << " "; cout << endl;
  delete[] dtomb;

  // vektor1 függvény csak feltölti
  vector<int> vektor1(elemszam);
  Feltolt(vektor1, tesztadat);
  for (int elem : vektor1)
    cout << elem << " "; cout << endl;

  // vektor2 függvény létrehozza és feltölti
  vector<int> vektor2;
  vektor2 = Letrehoz_es_Feltolt(elemszam, tesztadat);
  for (int elem : vektor2)
    cout << elem << " "; cout << endl;
  cin.get();
  return 0;
}
```

17. Egydimenziós tömbök rendezése

A szükséges fejállományok: <iostream>, < vector>, <algorithm>

```
bool Hasonlit(const int& a, const int& b) {
  return a > b;
}

int main() {
  const int elemszam = 10;

  // Statikus tömb
  int stomb[elemszam] = {7, 29, 10, 2, -9, 12, 11, 30, -4, 23};
  sort(begin(stomb), end(stomb));
  for (int elem : stomb) cout << elem << " "; cout << endl;
  // Összehasonlító függvény
  sort(stomb, stomb+elemszam, Hasonlit);
  for (int elem : stomb) cout << elem << " "; cout << endl;

  // Dinamikus tömb
  int *dtomb = new int[elemszam] {7, 29, 10, 2, -9, 12, 11, 30, -4, 23};
  sort(dtomb, dtomb+elemszam);
  for (int i=0; i<elemszam; i++) cout << dtomb[i] << " "; cout << endl;
  // Összehasonlító függvény
  stable_sort(dtomb, dtomb+elemszam, Hasonlit);
  for (int i=0; i<elemszam; i++) cout << dtomb[i] << " "; cout << endl;
  delete[] dtomb;

  // vektor
  vector<int> vektor {7, 29, 10, 2, -9, 12, 11, 30, -4, 23};
  sort(begin(vektor), end(vektor));
  for (int elem : vektor) cout << elem << " "; cout << endl;
  // Lambda-kifejezés
  stable_sort(begin(vektor), end(vektor), [](const int& a, const int& b) {return a>b;});
  for (int elem : vektor) cout << elem << " "; cout << endl;
  cin.get();
  return 0;
}
```