Wstęp do programowania w języku C

Przeszukiwanie z nawrotami - zagadka Einsteina
Obsługa wyjątków w C++

Marek Piotrów - Wykład 14

27 stycznia 2022

Zagadka przypisywana Einsteinowi I

1. Pięć kolorowych domów stoi w rzędzie. Każdy dom ma właściciela, który posiada zwierzę, ulubiony gatunek papierosów i ulubiony napój.
2. Anglik mieszka w czerwonym domu.
3. Hiszpan ma psa.
4. W zielonym domu piją kawę.
5. Ukrainiec pije herbatę.
6. Zielony dom sąsiaduje z białym.
7. Właściciel węża pali Winstony.
8. W żółtym domu palą Koole.
9. W domu znajdującym się pośrodku piją mleko.
10. Norweg mieszka w pierwszym domu od lewej.

Zagadka przypisywana Einsteinowi II

11. Palacz Chesterfieldów jest sąsiadem właściciela lisa.
12. W domu sąsiadującym z domem właściciela konia palą Koole.
13. Palacz Lucky Strików pije sok.
14. Japończyk pali Kenty.
15. Norweg sąsiaduje z niebieskim domem.

Pytanie: Kto jest właścicielem słonia, a kto pije wódkę?

zagadka.c I

```
#include<stdio.h>
#include<stdbool.h>

/*****************************************************************
Zagadka przypisywana Einsteinowi:

1.  Pięć kolorowych domów stoi w rzędzie. Każdy dom ma właściciela,
    który posiada zwierzę, ulubiony gatunek papierosów i ulubiony napój.
2.  Anglik mieszka w czerwonym domu.
3.  Hiszpan ma psa.
4.  W zielonym domu piją kawę.
5.  Ukrainiec pije herbatę.
6.  Zielony dom sąsiaduje z białym.
7.  Właściciel węża pali Winstony.
8.  W żółtym domu palą Koole.
9.  W domu znajdującym się pośrodku piją mleko.
10. Norweg mieszka w pierwszym domu od lewej.
11. Palacz Chesterfieldów jest sąsiadem właściciela lisa.
12. W domu sąsiadującym z domem właściciela konia palą Koole.
13. Palacz Lucky Strików pije sok.
14. Japończyk pali Kenty.
15. Norweg sąsiaduje z niebieskim domem.

16. Kto jest właścicielem słonia, a kto pije wódkę?
*****************************************************************/

// Autor programu: Marek Piotrów 30-01-2019
```

zagadka.c II

```
typedef enum os {Anglik, Hiszpan, Ukrainiec, Norweg, Japonczyk}        Osoba;
typedef enum ko {czerwony, zielony, bialy, zolty, niebieski}           Kolor;
typedef enum zw {pies, waz, lis, kon, slon}                            Zwierzak;
typedef enum pa {winstony, koole, chesterfieldy, luckyStriky, kenty} Papierosy;
typedef enum na {kawa, herbata, mleko, sok, wodka}                     Napoj;
typedef struct dom { Osoba o; Kolor k; Zwierzak z; Papierosy p; Napoj n; } Dom;
typedef Dom Ulica[5];

static char *strOsoba[] = {"Anglik", "Hiszpan", "Ukrainiec", "Norweg", u8"japończyk"};
static char *strKolor[] = {"czerwony", "zielony", u8"biały", u8"żółty", "niebieski"};
static char *strZwierzak[] = {"psa", u8"węża", "lisa", "konia", u8"słonia"};
static char *strPapierosy[] = {"winstony", "koole", "chesterfieldy", "luckyStriky", "kenty"};
static char *strNapoj[] = {u8"kawę", u8"herbatę", "mleko", "sok", u8"wódkę"};

void wypiszDom(Dom d)
{
  printf("Dom %s, mieszka: %s, ma %s, pali %s i pije %s\n",
        strKolor[d.k], strOsoba[d.o], strZwierzak[d.z], strPapierosy[d.p], strNapoj[d.n]);
}

void wypiszUlice(Ulica u)
{
  static int nr = 0;
  printf("Ulica[%d]:\n", nr++);
  for (int i = 0; i < 5; i++) printf("      "), wypiszDom(u[i]);
  putchar('\n');
```

zagadka.c III

```
}

void stworzUlice(Ulica u, Osoba ol[5], Kolor kl[5], Zwierzak zl[5], Papierosy pl[5], Napoj nl[5])
{
  for (int i = 0; i < 5; i++)
    u[i].o = ol[i],  u[i].k = kl[i],  u[i].z = zl[i],  u[i].p = pl[i],  u[i].n = nl[i];
}

bool wJednym(int A, int tabA[5], int B, int tabB[5])
{
  for (int i = 0; i < 5; i++)
    if (A == tabA[i] && B == tabB[i]) return true;
  return false;
}

bool wSasiednich(int A, int tabA[5], int B, int tabB[5])
{
  for (int i = 0; i < 5; i++)
    if (A == tabA[i] && ((i > 0 && B == tabB[i-1]) || (i < 4 && B == tabB[i+1])))
       return true;
  return false;
}
```

zagadka.c IV

```
static inline bool w2(Osoba ol[5], Kolor kl[5]) {
  return wJednym(Anglik, (int*)ol, czerwony, (int*)kl); }
static inline bool w3(Osoba ol[5], Zwierzak zl[5]) {
  return wJednym(Hiszpan, (int*)ol, pies, (int*)zl); }
static inline bool w4(Kolor kl[5], Napoj nl[5]) {
  return wJednym(zielony, (int*)kl, kawa, (int*)nl); }
static inline bool w5(Osoba ol[5], Napoj nl[5]) {
  return wJednym(Ukrainiec, (int*)ol, herbata, (int*)nl); }
static inline bool w6(Kolor kl[5]) {
  return wSasiednich(zielony, (int*)kl, bialy, (int*)kl); }
static inline bool w7(Papierosy pl[5], Zwierzak zl[5]) {
  return wJednym(winstony, (int*)pl, waz, (int*)zl); }
static inline bool w8(Kolor kl[5], Papierosy pl[5]) {
  return wJednym(zolty, (int*)kl, koole, (int*)pl); }
static inline bool w9(Napoj nl[5]) {  return mleko == nl[2]; }
static inline bool w10(Osoba ol[5]) {  return Norweg == ol[0] ; }
static inline bool w11(Papierosy pl[5], Zwierzak zl[5]) {
  return wSasiednich(chesterfieldy, (int*)pl, lis, (int*)zl); }
static inline bool w12(Papierosy pl[5], Zwierzak zl[5]) {
  return wSasiednich(koole, (int*)pl, kon , (int*)zl); }
static inline bool w13(Napoj nl[5], Papierosy pl[5]) {
  return wJednym(sok, (int*)nl, luckyStriky , (int*)pl); }
static inline bool w14(Osoba ol[5], Papierosy pl[5]) {
  return wJednym(Japonczyk, (int*)ol, kenty, (int*)pl); }
static inline bool w15(Osoba ol[5], Kolor kl[5]) {
  return wSasiednich(Norweg, (int*)ol, niebieski, (int*)kl); }
```

zagadka.c V

```
bool nastPerm(int n, int p[])
{
  if (n <= 1) return false;
  if (nastPerm(n-1, p+1)) return true;
  if (p[0] > p[1]) return false;
  for (int i = n-1; i > 0; i--)
    if (p[i] > p[0]) {
      int x = p[0]; p[0] = p[i]; p[i] = x;      // zamień p[0] z p[i]
      for (int j = 1, k = n-1; j < k; j++, k--) // odwróć ciąg p[1] .. p[n-1]
        x = p[j], p[j] = p[k], p[k] = x;
      break;
    }
  return true;
}

void przejrzyjOsoby(void);
void przejrzyjKolory   (Osoba ol[5]);
void przejrzyjNapoje   (Osoba ol[5], Kolor kl[5]);
void przejrzyjPapierosy(Osoba ol[5], Kolor kl[5], Napoj nl[5]);
void przejrzyjZwierzaki(Osoba ol[5], Kolor kl[5], Napoj nl[5], Papierosy pl[5]);

int main(void)
{
  przejrzyjOsoby();
  return 0;
```

zagadka.c VI

```
}
```

zagadka.c VII

```
void przejrzyjOsoby(void)
{
  Osoba ol[] =  {Anglik, Hiszpan, Ukrainiec, Norweg, Japonczyk};
  do {
    if (w10(ol)) przejrzyjKolory(ol);
  } while (nastPerm(5, (int*)ol));
}

void przejrzyjKolory(Osoba ol[5])
{
  Kolor kl[] = {czerwony, zielony, bialy, zolty, niebieski};
  do {
    if (w6(kl) && w2(ol, kl) && w15(ol, kl)) przejrzyjNapoje(ol, kl);
  } while (nastPerm(5, (int*)kl));
}

void przejrzyjNapoje(Osoba ol[5], Kolor kl[5])
{
  Napoj nl[] = {kawa, herbata, mleko, sok, wodka};
  do {
    if (w9(nl) && w4(kl, nl) && w5(ol, nl)) przejrzyjPapierosy(ol, kl, nl);
  } while (nastPerm(5, (int*)nl));
}
```

zagadka.c VIII

```
void przejrzyjPapierosy(Osoba ol[5], Kolor kl[5], Napoj nl[5])
{
  Papierosy pl[] = {winstony, koole, chesterfieldy, luckyStriky, kenty};
  do {
    if (w8(kl, pl) && w13(nl, pl) && w14(ol, pl)) przejrzyjZwierzaki(ol, kl, nl, pl);
  } while (nastPerm(5, (int*)pl));
}

void przejrzyjZwierzaki(Osoba ol[5], Kolor kl[5], Napoj nl[5], Papierosy pl[5])
{
  Zwierzak zl[] = {pies, waz, lis, kon, slon};
  do {
    if (w3(ol, zl) && w7(pl, zl) && w11(pl, zl) && w12(pl, zl)) {
      Ulica u;
      stworzUlice(u, ol, kl, zl, pl, nl);
      wypiszUlice(u);
    }
  } while (nastPerm(5, (int*)zl));
}
```

Rzucanie i przechwytywanie wyjątków

```
#include <iostream>

using namespace std;

int main(void) {
   try
   {
     bool sytuacja_wyjatkowa=true;
     const int opis_wyjatku=1;

     if (sytuacja_wyjatkowa)
        throw opis_wyjatku;

   }
   catch (int e)
   {
      cout << "Wystapil wyjatek o numerze " << e << endl;
   }
   return 0;
}
```

Przechwytywanie wielu wyjątków

```
try {

   // tu jakis kod

}
catch (int n) {
   cout<<"Wyjatek typu int o wartosci: "<<n;
}
catch (char c) {
   cout << "Wyjatek typu char o wartosci: "<<c;
}
catch (...) {
   cout << "Inny wyjatek";
}
```

Częściowa obsługa wyjątku

```
try {
   try {

         // tu jakis kod

   }
   catch (int n) {
         throw;
   }
}
catch (...) {
   cout << "Inny wyjatek";
}
```

Deklaracja rzucanych wyjątków

```
int funkcja1 (int n) throw (int);   // moze rzucic wyjatek tylko typu int

int funkcja2 (int n) throw();       // nie moze rzucic zadnym wyjatkiem

int funkcja3 (int n);               // moze rzucic dowolnym wyjatkiem
```

Klasa ułamków z rzucaniem wyjątków I

```
#include <iostream>
#include <exception>

using namespace std;

class UWyjatek: public exception {
   const char *tekst;
public:
   UWyjatek(const char *jaki=NULL): tekst(jaki) {}
   virtual const char *what() const throw() {
      return tekst;
   }
};

class Ulamek {
   long long licznik;
   long long mianownik;
private:
   long long nwd(long long n,long long m) throw()
   {
      for (long long r; m != 0; n=m,m=r) r=n%m;
      return n;
   }
   void skroc(void) throw(UWyjatek) {
      if (mianownik == 0) throw UWyjatek("Ulamek o mianowniku zero");
      if (mianownik < 0) { licznik=-licznik; mianownik=-mianownik; }
      long long int k = nwd((licznik >= 0 ? licznik : -licznik),mianownik);
```

Klasa ułamków z rzucaniem wyjątków II

```
        if (k > 1) {
            licznik/=k;
            mianownik/=k;
        }
    }
public:
    Ulamek(long long n=0,long long m=1) throw(UWyjatek):licznik(n),mianownik(m) {
        skroc();
    }

    bool operator== (const Ulamek &u) const {
        return licznik==u.licznik && mianownik==u.mianownik;
    }
    Ulamek operator+ (const Ulamek &u) const {
        return Ulamek(licznik*u.mianownik+mianownik*u.licznik,mianownik*u.mianownik);
    }
    Ulamek operator- (const Ulamek &u) const {
        return Ulamek(licznik*u.mianownik-mianownik*u.licznik,mianownik*u.mianownik);
    }
    Ulamek operator* (const Ulamek &u) const {
        return Ulamek(licznik*u.licznik,mianownik*u.mianownik);
    }
    Ulamek operator/ (const Ulamek &u) const {
        if (u == Ulamek(0)) throw UWyjatek("Dzielenie przez zero");
        return Ulamek(licznik*u.mianownik,mianownik*u.licznik);
    }
```

Klasa ułamków z rzucaniem wyjątków III

```
    double value(void) const {
        return static_cast<double>(licznik)/mianownik;
    }

    friend ostream & operator<< (ostream &os,Ulamek u) {
        os<<u.licznik;
        if (u.mianownik != 1) os<<'/'<<u.mianownik;
        return os;
    }
    friend istream & operator>> (istream &is,Ulamek &u) {
        long long n,m;
        char c;
        is>>n>>c>>m;
        if (c != '/') throw UWyjatek("Zly ulamek");
        u=Ulamek(n,m);
        return is;
    }
};
```

Test klasy ułamków z przechwytywaniem wyjątków I

```
#include <iostream>
#include <iomanip>
#include "ulamek_z_obsluga_wyjatkow.h"

using namespace std;

Ulamek pi_drugich(int n,int a)
{ // wzor Newtona Pi/2 = 1 + 1/3 * (1 + 2/5 * (1 + 3/7 * (1 + 4/9 * ( ... ))))
   if (n > 0)
     return Ulamek(a,2*a+1) * pi_drugich(n-1,a+1) + 1;
   else
      return Ulamek(3,2);
}

int main(int argc,char *argv[])
{
   try {
      Ulamek u1,u2,u3;

      cin>>u1>>u2; u3=u1*u2;
      cout<<u1<<" + "<<u2<<" = "<<u1+u2<<endl;
      cout<<u1<<" - "<<u2<<" = "<<u1-u2<<endl;
      cout<<u1<<" * "<<u2<<" = "<<u3<<endl;
      cout<<u1<<" + "<<u1<<" / "<<u2<<" = "<<u1+u1/u2<<endl;

      Ulamek pi=pi_drugich(22,1)*2;
```

Test klasy ułamków z przechwytywaniem wyjątków II

```
    cout<<"Pi = "<<pi<<" = "<<setprecision(10)<<pi.value()<<endl;
  }
  catch (exception &e) {
    cerr<<argv[0]<<": BLAD: "<<e.what()<<endl;
    return 1;
  }
  return 0;
}
```

Wyjątki - kilka ogólnych zasad

- Wszystkie wyjątki rzucane przez program powinny być obsługiwane.
- Jeśli program używa standardowej biblioteki C++, powinien obsługiwać rzucane przez nią wyjątki.
- Procedura obsługi wyjątku jest wybierana na podstawie typu rzucanego wyjątku.
- Kolejne procedury obsługi wyjątków powinny obsługiwać typy od najbardziej szczególnych do najbardziej ogólnych.
- Specyfikacja wyjątków przy deklaracjach funkcji i metod ułatwia zrozumienie i kontrolę systemu obsługi wyjątków.