Wstęp do programowania w języku C

Marek Piotrów - Wykład 14
Automatyczna kompilacja - Makefile
Nowe elementy w C++:
przeciążanie operatorów, obsługa wyjątków, wzorce klas

4 lutego 2015

Automatyzacja kompilacji projektów - Makefile

- Standardowy program `make` automatyzuje kompilację projektów.
- `make` czyta skrypt(y) `Makefile` i na ich podstawie ustala, które pliki wymagają kompilacji.
- Następnie kompiluje je według podanych w `Makefile` reguł.
- Po wprowadzeniu zmian do projektu kompilowane są tylko moduły, na które zmiana mogła mieć wpływ.
- Istnieje kilka wersji tego programu: np. GNU make, czy Miscrosoft make.
- `Makefile` zawiera zbiór reguł dla `make` opisujący zależności między plikami i ruguły kompilacji.
- Narzędzie `cbp2make` automatycznie tworzy `Makefile` dla projektu Code::Blocks.

Trywialny Makefile

```
szesciokat : punkty.c szesciokat.c punkty.h
	gcc -std=c99 -Wall -Wextra -Werror punkty.c szesciokat.c -o szesciokat -lm
```

Najprostszy Makefile

```
# najprostszy makefile dla programu szesciokat

szesciokat: punkty.o szesciokat.o
	gcc -std=c99 -Wall -Wextra -Werror punkty.o szesciokat.o -o szesciokat -lm

punkty.o: punkty.c punkty.h
	gcc -std=c99 -Wall -Wextra -Werror -c punkty.c -o punkty.o

szesciokat.o: szesciokat.c punkty.h
	gcc -std=c99 -Wall -Wextra -Werror -c szesciokat.c -o szesciokat.o
```

Prosty Makefile

```
# prosty makefile dla programu: szesciokat

CC=gcc
CFLAGS= -std=c99 -Wall -Wextra -Werror
# ogolne flagi kompilacji dla modulow

LDFLAGS= -lm
# ogolne flagi konsolidacji programu

DEPS = punkty.h
NAME = szesciokat

$(NAME): punkty.o szesciokat.o
	$(CC) $(CFLAGS) punkty.o szesciokat.o -o $(NAME) $(LDFLAGS)

punkty.o: punkty.c $(DEPS)
	$(CC) -c $(CFLAGS) punkty.c -o punkty.o

szesciokat.o: szesciokat.c $(DEPS)
	$(CC) -c $(CFLAGS) szesciokat.c -o szesciokat.o
```

Standardowy Makefile

```
# standardowy makefile dla programu: szesciokat

CC=gcc
CFLAGS = -std=c99 -Wall -Wextra -Werror
LFLAGS = -lm

NAME = szesciokat
# nazwa programu wynikowego
SRC =  szesciokat.c punkty.c
DEPS = punkty.h
# nazwy wszystkich plikow zrodlowych
OBJS = szesciokat.o punkty.o
# nazwy wszystkich modulow

YOU : $(SRC) $(NAME)

$(NAME): $(OBJS)
	$(CC) $(CFLAGS) $(OBJS) -o $(NAME) $(LFLAGS)

%.o: %.c $(DEPS)
	$(CC) $(CFLAGS) -c -o $@ $<
```

Makefile dla pracy magisterskiej I

```
OPTIONS = -DNDEBUG
# OPTIONS = -DPOOR_LIBS -DNDEBUG
# Use the above line if you have problems with compilation due to
# missing declarations. This enables including of "fixes.h", where you
# can place or include what is missing.
# when finished, uncomment -DNDEBUG to remove debugging code

CFLAGS = -s -funsigned-char -O2 # -fno-exceptions
# when finished, add -O2 to CFLAGS
# -fno-exceptions is for gcc 2.8.x
# (I don't use exceptions -> executable will be smaller)
LFLAGS = -lm # -lstdc++

SRC = euphoria.l euphoria.y symtab.cc symtab.h fixes.h init.cc code.h code.cc \
errors.h seq.cc seq.h
OBJS = symtab.o init.o code.o euphoria.o seq.o
NAME = euphoria

YOU : $(SRC) $(NAME)

$(NAME) : $(OBJS)
	gcc $(CFLAGS) $(OPTIONS) -o $(NAME) $(OBJS) $(LFLAGS)

lex.yy.c : euphoria.l
	flex -s euphoria.l
```

Makefile dla pracy magisterskiej II

```
euphoria.tab.c : euphoria.y
	bison euphoria.y

euphoria.o : euphoria.tab.c lex.yy.c
	gcc -c $(CFLAGS) $(OPTIONS) -x c++ $< -o $@

%.o : %.cc
	gcc -c $(CFLAGS) $(OPTIONS) $< -o $@
```

Klasa `Ulamek` I

```
#include <iostream>

using namespace std;

class Ulamek {
   long long licznik;
   long long mianownik;
private:
   long long nwd(long long n,long long m)
   {
      for (long long r; m != 0; n=m,m=r) r=n%m;
      return n;
   }
   void skroc(void) {
      if (mianownik < 0) { licznik=-licznik; mianownik=-mianownik; }
      long long int k = nwd((licznik >= 0 ? licznik : -licznik),mianownik);
      if (k > 1) {
         licznik/=k;
         mianownik/=k;
      }
   }
public:
   Ulamek(long long n=0,long long m=1):licznik(n),mianownik(m) {
      skroc();
   }
   // Konstruktor kopiujący - bedzie wygenerowany automatycznie
```

Klasa `Ulamek` II

```
// Ulamek(const Ulamek &u):licznik(u.licznik),mianownik(u.mianownik) { }

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 {
   return Ulamek(licznik*u.mianownik,mianownik*u.licznik);
}
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;
}
```

Klasa `Ulamek` III

```
    friend istream & operator>> (istream &is,Ulamek &u) {
        long long n,m;
        char c;
        is>>n>>c>>m;
        u=Ulamek(n,m);
        return is;
    }
};
```

Program testujący klasę

```
#include <iostream>
#include <iomanip>
#include "ulamek.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(1)+Ulamek(a,2*a+1)*pi_drugich(n-1,a+1);
   else
      return Ulamek(3,2);
}

int main(void)
{
   Ulamek u1,u2,u3;
   Ulamek pi=pi_drugich(22,1)*Ulamek(2);

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

   cout<<"Pi = "<<pi<<" = "<<setprecision(10)<<pi.value()<<endl;
   return 0;
}
```

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; }
```

Klasa ułamków z rzucaniem wyjątków II

```
        long long int k = nwd((licznik >= 0 ? licznik : -licznik),mianownik);
        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.

Wzorzec klasy kolejka

```
#include <cstddef>
using namespace std;

template <class Typ> class kolejka
{
private:
   class element
   {
   friend class kolejka;
   private:
      Typ info;
      element *nastepny;
   public:
      element(element *&list, const Typ &i);
      ~element(void);
   public:
      const Typ wartosc(void) { return(info); }
   };
   element *lista;
public:
   kolejka(void);
   ~kolejka(void);
public:
   bool pusta(void) { return lista==NULL; }
   void wstaw(const Typ &i);
   void usun(void);
   const Typ podaj(void);
};
```

Kolejka - konstruktory i destruktory

```
template <class Typ>
kolejka<Typ>::element::element(element *&list, const Typ &i):info(i)
{
   if (list != NULL) {
      nastepny=list->nastepny;
      list->nastepny=this;
   }
   else nastepny=this;
   list=this;
}

template <class Typ>
kolejka<Typ>::element::~element(void)
{

}

template <class Typ>
kolejka<Typ>::kolejka(void)
{
   lista=NULL;
}

template <class Typ>
kolejka<Typ>::~kolejka(void)
{
   while (lista != NULL) usun();
}
```

Kolejka - operacje

```
template <class Typ>
void kolejka<Typ>::wstaw(const Typ &i)
{
   new element(lista,i);
}

template <class Typ>
void kolejka<Typ>::usun(void)
{
   if (lista == NULL)
      return;
   if (lista->nastepny == lista)
   {
      delete lista;
      lista=NULL;
      return;
   }
   element *pom;
   pom=lista->nastepny;
   lista->nastepny=pom->nastepny;
   delete pom;
}

template <class Typ>
const Typ kolejka<Typ>::podaj(void)
{
   if (lista == NULL) throw 2;
   return lista->nastepny->wartosc();
}
```

Kolejka - test wzorca klasy

```
#include <iostream>
#include <string>
#include "kolejka.h"

using namespace std;

int main(void)
{
   kolejka<int> kint;
   kolejka<string> kstring;

   cout<<"=== Podaj liczby do kolejki (0-koniec) ==="<<endl;
   for (int liczba; cin>>liczba,liczba != 0; ) kint.wstaw(liczba);
   cout<<"=== Zawartosc kolejki liczb ==="<<endl;
   for ( ; !kint.pusta(); kint.usun()) cout<<kint.podaj()<<' ';
   cin.ignore(128,'\n');

   cout<<endl<<"=== Podaj napisy do kolejki (pusty-koniec) ==="<<endl;
   for (char buf[128]; cin.getline(buf,sizeof(buf)),buf[0] != 0; )
      kstring.wstaw(string(buf));
   cout<<"=== Zawartosc kolejki napisow ==="<<endl;
   for ( ; !kstring.pusta(); kstring.usun())
     cout<<'\"'<<kstring.podaj()<<"\" ";
   cout<<endl<<"====== Koniec ========"<<endl;
   return 0;
}
```

Kolejka - test wzorców klas z biblioteki STL

```
#include <iostream>
#include <string>
#include <list>
#include <queue>

using namespace std;

int main()
{
   list<int> kint;
   queue<string> kstring;
   queue<pair<int,string> > pary;

   cout<<"=== Podaj liczby do listy (0-koniec) ==="<<endl;
   for (int liczba; cin>>liczba,liczba != 0; ) kint.push_back(liczba);
   cout<<"=== Zawartosc listy liczb ==="<<endl;
   for ( ; !kint.empty(); kint.pop_front()) cout<<kint.front()<<' ';
   cin.ignore(128,'\n');

   cout<<endl<<"=== Podaj napisy do kolejki (pusty-koniec) ==="<<endl;
   for (char buf[128]; cin.getline(buf,sizeof(buf)),buf[0] != 0; )
      kstring.push(string(buf));
   cout<<"=== Zawartosc kolejki napisow ==="<<endl;
   for (int i=1 ; !kstring.empty(); i++,kstring.pop()) {
     pary.push(pair<int,string>(i,kstring.front()));
     cout<<'\"'<<pary.back().second<<"\" ";
   }
   cout<<endl<<"====== Koniec ========"<<endl;
}
```

Dziękuję za uwagę

Dziękuję wszystkim za uwagę.