Beata Jankowska,
Instytut Automatyki, Robotyki i Inżynierii Informatycznej

IV. Obsługa błędów i obsługa wyjątków. Implementacja wyjątków w języku C++

1. Błędy w programie

1.1. Błędy występujące w programie można podzielić na dwie podstawowe kategorie:
- błędy składniowe i
- błędy semantyczne.

1.2. Błędy składniowe, zwane też **błędami czasu kompilacji** powstają na skutek nieprawidłowego zapisu tekstu programu. Kompilator wykrywa je na etapie analizy programu, w fazach:

- analizy leksykalnej – błędy wynikające z niepoprawnego zapisu jednostek leksykalnych (słów kluczowych, literałów, identyfikatorów, operatorów, separatorów, itd.) w programie,
- analizy składniowej – błędy związane z nieprawidłowym zapisem fraz składniowych (dyrektyw, deklaracji, wyrażeń, instrukcji, itd.) w programie,
- analizy zależności kontekstowych – błędy wynikające z niewłaściwego użycia identyfikatorów (niedotrzymanie wymogu unikatowości identyfikatora, użycie identyfikatora poza zakresem jego widoczności), niewłaściwego kontekstu wystąpienia instrukcji, lub – szeroko pojętej – niezgodności typów argumentów w wyrażeniu.

1.3. Większość wykrytych na etapie analizy błędów składniowych wiąże się z koniecznością przerwania procesu kompilacji i koniecznością sygnalizacji miejsca i rodzaju błędów w programie. Tylko w nielicznych wypadkach kompilator samodzielnie usuwa błąd z programu. Podjęcie decyzji o **neutralizacji błędu** wymaga w każdym wypadku przyjęcia założenia na temat zamierzonej jednostki leksykalnej, zamierzonej frazy składniowej lub zamierzonego typu argumentu w wyrażeniu. Do najbezpieczniejszych

przekształceń programu realizowanych automatycznie w procesie jego kompilacji należą niejawne konwersje typów, zwane koercjami.

1.4. Błędy semantyczne wykrywane są dopiero w procesie uruchamiania programu. Na tym etapie może się okazać, że pozornie poprawny program ma znaczenie różne od zamierzonego, tzn. nie spełnia zbioru przyjętych niezmienników. Błąd semantyczny może być konsekwencją niepoprawnej implementacji dobrze skonstruowanego algorytmu, lub – co gorsza – efektem niepoprawnej konstrukcji samego algorytmu. Wystąpienie błędu semantycznego oznacza więc często konieczność rekonstrukcji całego programu.

1.5. Wśród błędów semantycznych można jednak spotkać i inne, mniej poważne. Zdarza się, że program działa poprawnie dla wszystkich reprezentatywnych zestawów danych wejściowych i niepoprawnie dla pewnych nielicznych zestawów danych brzegowych. W tych ostatnich wypadkach błędy polegają, między innymi, na przepełnianiu stosów maszynowych lub programowych, indeksowaniu tablic indeksami spoza ich zakresu, wyznaczaniu (w efekcie obliczeń) wartości spoza przedziałów implementujących typy liczbowe, czy odwołaniach do nieistniejących obiektów lub innych struktur danych poprzez zmienne wskaźnikowe (tzw. wiszące referencje). Wymienione błędy są **błędami czasu wykonania.** Nie zawsze da się ich uniknąć, można natomiast zabezpieczyć się przed skutkami ich wystąpienia.

2. Wyjątki – zgłaszanie i obsługa

2.1. Stanem wyjątkowym (**wyjątkiem**) nazywamy wystąpienie błędu lub innego zdarzenia, które przerywa normalne **wykonywanie programu**. Właściwa obsługa stanów wyjątkowych pozwala na podniesienie niezawodności programu.

2.2. Specyfika obsługi wyjątków w językach obiektowych polega na oddzieleniu etapu wykrywania wyjątków (czyli anomalii programowych) od etapu ich obsługi. Mechanizm wyjątku składa się więc z:

- rozpoznania stanu wyjątkowego,
- instrukcji zgłoszenia wyjątku oraz
- obsługi wyjątku.

Rozpoznanie stanu wyjątkowego może mieć charakter sprzętowy lub programowy. W pierwszym wypadku następuje automatyczne zgłoszenie wyjątku o charakterze standardowym. Programowe rozpoznanie stanu wyjątkowego ma zwykle postać instrukcji warunkowej, z testem okoliczności wystąpienia tego stanu i warunkowym **zgłoszeniem wyjątku** (standardowego lub niestandardowego). Sekcja **obsługi wyjątku** to ciąg instrukcji wykonywanych w sytuacji wyjątkowej i mających na celu jej "rozwiązanie".

2.3. Rozpoznanie stanu wyjątkowego może być przybliżone lub precyzyjne. W najgorszym wypadku ma miejsce samo stwierdzenie błędu, bez dalszej jego klasyfikacji. W lepszej sytuacji następuje rozpoznanie błędu z dokładnością do jego klasy, w najlepszej – jednoznaczne określenie błędu.

3. Wyjątki w programach języka C++

3.1. W języku C++ przewidziano następujące mechanizmy do implementacji wyjątków o charakterze programowym:

- instrukcję zgłoszenia wyjątku,
- mechanizm obsługi wyjątku, obejmujący:
  - oznakowanie fragmentu programu, z którego pochodzi potencjalne zgłoszenie wyjątku (w dowolnym procesie obliczeniowym),
  - kod do obsługi wyjątku zgłoszonego (jawnie lub niejawnie) przez instrukcję pochodzącą z pewnego oznakowanego fragmentu programu.

3.2. Instrukcja zgłoszenia wyjątku ma postać:

`throw;` lub

`throw` *wyrazenie*`;`

gdzie *wyrazenie* jest dowolnym, zbudowanym poprawnie w C++ wyrażeniem, w szczególności – obiektem pewnej klasy. Przy założeniu poniższych deklaracji i definicji:

```
//pomocnicza definicja typu Lista
struct Lista
{ int elem;
  shared_ptr<Lista> nast; };

//deklaracje zmiennych i definicje funkcji
int p1;
float* wsk_zm2;
char* zm3;
string tab;
shared_ptr<Lista> wsk_zs;
int* funct1(string slowo, int ind) {/*...*/};
double funct2() {/*...*/};

//definicje klas
class Sygnal_bledu {};
class Obsluga_stosu
{
 public:
  shared_ptr<Lista> stos;
  Obsluga_stosu(shared_ptr<Lista> st) : stos(st) {}
  //definicje pozostałych metod klasy
};

//deklaracja zmiennej
Obsluga_stosu *wsk_Obs_stosu;
```

można zgłosić (w zakresie widoczności powyższych zmiennych, funkcji i klas) wyjątki:

```
throw 0;
throw p1;
throw 300*(p1-2);
throw *wsk_zm2;
throw zm3;
throw funct1(tab,30);
throw funct2();

//utworzenie listy wsk_zs
//inicjalizacja obiektu klasy Obsluga_stosu
//wsk_Obs_stosu=new Obsluga_stosu(wsk_zs);
throw wsk_Obs_stosu;

//deklaracja zmiennej wskaznikowej st
//shared_ptr<Lista> st;
throw st→elem;
```

3.3. Zgłoszenie wyjątku jest instrukcją. W związku z tym, może wystąpić jedynie w obrębie bloku definiującego pewną funkcję programu (w szczególności – funkcję `main`). Oto możliwy kontekst użycia kilku spośród wymienionych zgłoszeń wyjątków:

```
int Fun_z_wyj1(int p1)
//funkcja realizuje obliczenia numeryczne
{ if (p1<7) return -1;
  else
     if (p1==7) {throw 0;}
     else
         if (p1>9000) {throw p1;}
         else return (300*(p1-2)*300*(p1-2))%(p1-7);}
```

```
void Fun_z_wyj2(shared_ptr<Lista> wsk_ps, int licz)
//funkcja zdejmuje licz>0 elementow ze stosu zewn.
//wsk_zs, po czym umieszcza na nim wszystkie
//elementy ze stosu-parametru wsk_ps
{ shared_ptr<Lista> st;
  wsk_Obs_stosu=new Obsluga_stosu(wsk_zs);
  if (wsk_Obs_stosu→stos == nullptr)
      throw wsk_Obs_stosu;
  else { st = wsk_Obs_stosu→stos;
         for(int i=1; i<licz; i++) {
            if(st.get()→nast == nullptr)
              { throw st→elem;
                break;}
            else {/*…*/;} //usuniecie kolejnego,
                          //i-tego elementu z wewn.
                          //stosu obiektu klasowego
         };
         //usuniecie licz-tego elementu
         //z wewn. stosu obiektu wsk_Obs_stosu,
         //po czym wprowadzenie tam elementow
         //z wsk_ps...
         }
  wsk_zs = wsk_Obs_stosu→stos;
}

//przykład uzycia Fun_z_wyj2
//...
shared_ptr<Lista> wsk_ps;
//utworzenie listy wsk_ps o 30 elementach typu
//calkowitego
Fun_z_wyj2(wsk_ps,30);
//...
```

3.4. Obszar potencjalnego zgłoszenia wyjątku znakuje się w programie przy użyciu konstrukcji składniowej w postaci:

`try` {*blok*},

w której *blok* oznacza ciąg deklaracji i instrukcji zamieszczonych w obrębie dowolnej funkcji programu.

Przy uwzględnieniu przytoczonych definicji i deklaracji, można zastosować następujące oznakowanie obszarów potencjalnej aktywacji wyjątków w programie:

```
int Przyk_fun(int par)
{//definicja funkcji Przyk_fun
 //...
 try
 {switch (Fun_z_wyj1(par))
  {
   case -1 : printf("ujemny mianownik");break;
   case 0  : printf("podzielnosc");break;
   default : printf("brak podzielnosci");break;
  }
  //...
  return 0;
 }
 //dalsze instrukcje funkcji Przyk_fun
 return 1;
}//koniec definicji funkcji Przyk_fun
```

```
void Fun_z_wyj2(shared_ptr<Lista> wsk_pars, int licz)
//funkcja zdejmuje licz>0 elementow ze stosu
//globalnego wsk_zs, po czym umieszcza na nim
//wszystkie elementy ze stosu wsk_pars
{shared_ptr<Lista> st;
 wsk_Obs_stosu=new Obsluga_stosu(wsk_zs);
 try {
      if (wsk_Obs_stosu→stos==nullptr)
         {throw wsk_Obs_stosu;}
      else { st = wsk_Obs_stosu→stos;
             for(int i=1; i<licz; i++)
                {
                  if(st.get()→nast == nullptr)
                     {throw st.get()→elem;}
                  else {/*...*/;}
                  //usuniecie
                  //kolejnego, i-tego elementu
                  //ze stosu wewnętrznego
                }
             //usuniecie licz-tego elementu
             //z wewn. stosu obiektu
             //wsk_Obs_stosu, po czym
             //wprowadzenie tam elementow
             //ze wsk_pars...
           }
     }
 wsk_zs = wsk_Obs_stosu→stos;
}
```

```
void main()
{//definicja funkcji main
 string tab;
 //...
 char c;
 int i=0;
 shared_ptr<Lista> wsk_poms = nullptr;
 shared_ptr<Lista> wsk_poms1;
 c=tab.at(i); i++;
 while (c!='a')
  { wsk_poms1 = make_shared<Lista>;
    wsk_poms1.get()→elem = int(c);
    wsk_poms1.get()→nast = wsk_poms;
    wsk_poms = wsk_poms1;
    c= tab.at(i);
    i++;
  }
 //przyjmujemy, ze na liscie wsk_poms
 //znajduje się co najmniej jeden znak
 try
 { Fun_z_wyj2(wsk_poms,i-1);
 }
 //dalsze instrukcje funkcji main
}//koniec definicji funkcji main
```

3.5. Obsługa wyjątku odbywa się w sekcji obsługi, umieszczonej bezpośrednio za:

- oznakowanym obszarem potencjalnego zgłoszenia wyjątku lub
- inną sekcją obsługi wyjątku.

Sekcja obsługi wyjątku jest konstrukcją składniową w postaci:

`catch` (*deklaracja_wyjatku*) {*blok*} ,

w której *blok* oznacza dowolny ciąg deklaracji i instrukcji, zaś *deklaracja_wyjatku* – klauzulę w postaci:

(a) `...` (trójkropek),
(b) *id_typu_liczb/klasy* lub
(c) *id_typu param* ,

gdzie *id_typu_liczb/klasy* jest identyfikatorem typu liczbowego lub klasowego, *id_typu* – identyfikatorem dowolnego typu (w tym klasowego), a *param* – parametrem formalnym zgłoszenia wyjątku.

3.5.1. Sekcja obsługi z *deklaracja_wyjatku* w postaci (b) lub (c) przechwytuje wszystkie wyjątki typu *id_typu_liczb/klasy* lub *id_typu* (powyższy typ jest typem *wyrazenia* z instrukcji zgłoszenia wyjątku `throw` *wyrazenie*). Argument *param*, użyty dodatkowo w *deklaracji_wyjatku* (c), przekazuje do *bloku* wartość *wyrazenia*. Sekcja obsługi z *deklaracja_wyjatku* w postaci (a) przechwytuje wszystkie wyjątki niesklasyfikowane.

3.5.2. Z każdym obszarem zgłoszeń należy związać co najmniej jedną sekcję obsługi wyjątku. Liczba tych sekcji może być większa niż 1. W takim wypadku warto przestrzegać zasady wiązania z tym samym typem co najwyżej jednej sekcji obsługi. Porządek specyfikacji wszystkich sekcji postaci (b) lub (c) jest w zasadzie dowolny. Sekcję z *deklaracja_wyjatku* w postaci trójkropka warto jednak umieszczać na końcu, za wszystkimi pozostałymi sekcjami (w przeciwnym razie, program będzie zawierał kod martwy).

3.5.3. Kolejne instrukcje z oznakowanego obszaru zgłoszeń `try` są wykonywane dopóty, dopóki jedna z nich nie zgłosi wyjątku. W takiej sytuacji sterowanie jest przekazywane do właściwej sekcji obsługi `catch` znajdującej się za oznakowanym obszarem `try`. Po

wykonaniu wszystkich umieszczonych w niej instrukcji, sterowanie wraca do miejsca położonego bezpośrednio za oznakowanym obszarem `try`.

3.5.4. W wypadku, gdy za oznakowanym obszarem zgłoszeń `try` nie ma sekcji, która mogłaby obsłużyć zgłoszenie wyjątku z tego obszaru, następuje propagacja tego wyjątku wzdłuż dynamicznego łańcucha wywołań funkcji.

3.5.5. Wyjątki zgłaszane z obszaru `try` w obrębie funkcji `Przyk_fun` będą obsługiwane w sekcji umieszczonej za tym obszarem:

```
int Przyk_fun(int par)
{//definicja funkcji Przyk_fun
 //...
 try
 { switch (Fun_z_wyj1(par))
   {
    case -1 : printf("ujemny mianownik");break;
    case 0  : printf("podzielnosc");break;
    default : printf("brak podzielnosci");break;
   }
   //...
   return 0;
 }
 catch (int arg)
 { if (!arg) printf("dzielenie przez zero!!!\n");
   else printf("przekroczenie zakresu!!!\n");}
 catch (...)
 { printf("blad!!!\n");}
 //dalsze instrukcje funkcji Przyk_fun
 return 1;
}//koniec definicji funkcji Przyk_fun
```

3.5.6. Umieśćmy w ciele funkcji `Fun_z_wyj2` sekcję obsługi wyjątków typu `int`:

```
void Fun_z_wyj2(shared_ptr<Lista> wsk_pars, int licz)
//funkcja zdejmuje licz>0 elementow ze stosu
//globalnego wsk_zs, po czym umieszcza na nim
//wszystkie elementy ze stosu wsk_pars
{ shared_ptr<Lista> st;
  wsk_Obs_stosu=new Obsluga_stosu(wsk_zs);
  try {
       if (wsk_Obs_stosu→stos==nullptr)
          { throw wsk_Obs_stosu;}
       else { st = wsk_Obs_stosu→stos;
              for(int i=1; i<licz; i++)
                {
                  if(st.get()→nast == nullptr)
                     { throw st.get()→elem;}
                  else {/*...*/;}
                  //usuniecie
                  //kolejnego, i-tego elementu
                  //ze stosu wewnętrznego
                }
              //usuniecie licz-tego elementu
              //z wewn. stosu obiektu
              //wsk_Obs_stosu, po czym
              //wprowadzenie tam elementow
              //ze wsk_pars...
            }
     }
 wsk_zs = wsk_Obs_stosu→stos;
}
catch(int)
    { printf("nie da sie usunac ");
      printf("nakazanej liczby ");
      printf("elementow ze stosu wsk_ls\n");
      //kontynuacja dzialan – obliczenie
      //alternatywne
    }
wsk_ls = wsk_Obs_stosu→stos;
}
```

W nowej sytuacji zgłoszenie wyjątku `throw st→elem` zostanie obsłużone wewnątrz funkcji, w sekcji z deklaracją wyjątku `int`, zaś zgłoszenie wyjątku `throw wsk_Obs_stosu` zostanie przekazane do obsługi do funkcji `main`.
Funkcja `main` z niezbędną sekcją obsługi wyjątku przyjmuje ostatecznie postać:

```
void main()
{//definicja funkcji main
 string tab;
 //...
 char c;
 int i=0;
 shared_ptr<Lista> wsk_poms = nullptr;
 shared_ptr<Lista> wsk_poms1;
 c=tab.at(i); i++;
 while (c!='a')
  { wsk_poms1 = make_shared<Lista>;
    wsk_poms1.get()→elem = int(c);
    wsk_poms1.get()→nast = wsk_poms;
    wsk_poms = wsk_poms1;
    c= tab.at(i);
    i++;
  }
 //przyjmujemy, ze na liscie wsk_poms
 //znajduje się co najmniej jeden znak
 try
 { Fun_z_wyj2(wsk_poms,i-1);
 }
 catch(Obsługa_stosu *wsk_obiektu)
 { printf("stos globalny jest pusty;\n");
   printf("nie da się wykonac ");
   printf("nakazanych operacji\n");
   //przerwanie dzialan
   return;
 }
//dalsze instrukcje funkcji main
}//koniec definicji funkcji main
```