Laboratorium Podstaw Informatyki

1. Przebieg ćwiczenia laboratoryjnego

Obsługa wyjątków

Mechanizm wyjątków w języku C++ powstał w celu ulepszenia obsługi błędów programów. W wielu programach mogą pojawić się sytuacje, w których działanie kodu może zostać zakłócone. Jeżeli nasz program jest napisany bardzo dobrze, nie oznacza to, iż nie mogą wystąpić sytuacje, w których powstanie błąd (np. błąd dostępu do pliku, błąd przepełnienia pamięci, przekroczenie zakresu tablic itp.). Są to zazwyczaj sytuacje trudne do wychwycenia na poziomie pisania kodu programu i jego kompilacji. Z tego powodu warto zabezpieczyć te miejsca programu, w których może dojść do wystąpienia błędu a następnie poinformować o nim użytkownika.

Składnia wyjątków

Najprostsza obsługa wyjątków w C++ składa się z bloku **try**, instrukcji **throw** i bloku **catch**:

```
try
{
        //jakiś kod
        throw 20;
        //jaki kod który się nie wykona
}
catch (int)
{
        //kod obsługi wyjątku
}
//dalsza część programu
```

Wyjątki w programie nie przerywają działania całego kodu. Jest tu stosowana normalna procedura zakończenia bloku programu, tzn. wywoływane są destruktory obiektów lokalnych funkcji wywołanych z bloku try itp.

Przykład programu wykorzystujący mechanizm wyjątków:

```
void funkcja(void)
{
        int* tab;
        int except;
        tab = new int[1024];
        if (tab==NULL)
                {
                except = 40;
                throw except;
                }
}
int main(void)
{
  try
        {
        funkcja();
        }
  catch (int a)
        {
        cout << "Błąd " << a << endl;
        }
  catch (...)
         {
        cout << "Błąd niezdefiniowany !" << endl;
         }
 return 0;
}
```

Widać tutaj funkcję która próbuje zaalokować pamięć na potrzeby tablicy. Jeżeli alokacja pamięci się nie uda (wskaźnik będzie wskazywał na NULL), to rzucamy wyjątek typu int do którego przypisujemy wcześniej wartość 40. Następnie w głównej funkcji programu próbujemy (try) wykonać funkcję o której wiemy, że może nie zakończyć się poprawnie. Po bloku try występują blok (bloki) catch, które łapią rzucony wyjątek i zależnie od tego co złapią wyświetlają odpowiedni błąd na konsoli.

Uwagi praktyczne:

- Blok try powinien obejmować możliwie minimalną ilość kodu programu, co do którego wiemy, iż może wygenerować wyjątek.
- Bloków catch może być dowolnie dużo, zależnie od tego jakiego typu wyjątki mogą być rzucone. Ponieważ rzucać można dowolne obiekty (a szczególnie obiekty klasy

użytkownika), zatem bloki catch obowiązuje hierarchia wynikająca z dziedziczenia. Oznacza to iż bloki catch powinny łapać wyjątki od najbardziej szczegółowego do najbardziej ogólnego.

- Jako ostatni blok catch można umieścić blok catch (…) – z wielokropkiem. Jest to blok który łapie wszystkie wyjątki, ale nie ma możliwości identyfikacji jakiego konkretnie typu wyjątek został złapany.

Treść zadania laboratoryjnego:

1. Utwórz klasę **Wyjatek**, która będzie dostarczała obiektów do rzucania wyjątków. W jej wnętrzu zaimplementuj dwa pola prywatne: **int numer** oraz **string opis** służące przechowywaniu informacji o błędzie. Zdefiniuj konstruktor klasy wyjątek w taki sposób, aby podczas tworzenia obiektu klasy nadawał numer i opis poprzez stosowne argumenty. Dodatkowo zaimplementuj dwie metody **getNum()** oraz **getOpis()** odpowiedniego typu. Pierwsza ma zwracać numer wyjątku. Druga ma zwracać jego opis. Zaimplementuj w klasie wyjatek destruktor wypisujący na ekran informacje o ich uruchomieniu (np. „Jestem w destruktorze klasy wyjątek”).

2. Utwórz klasę **Stos**, która w sekcji prywatnej będzie miała wskaźnik **int* tablica**. Konstruktor klasy ma za zadanie utworzyć dynamicznie tablicę 10 elementów wykorzystując w tym celu wskaźnik z sekcji prywatnej. Destruktor klasy powinien zwolnić pamięć tablicy. Aby zaobserwować działanie konstruktora oraz destruktora dodaj w ciałach tych metod instrukcje wypisujące na ekran informacje o uruchomieniu danej metody (np. „Jestem w konstruktorze klasy stos”, „Jestem w destruktorze klasy stos”)

3. W klasie **Stos** zaimplementuj metodę **void push(int arg, int i)**, która będzie wprowadzała argument **arg** do **i**-tej komórki tablicy **tablica**. W przypadku próby przekroczenia rozmiaru tablicy (czyli dla i>9) metoda ma utworzyć wskaźnik **Wyjatek* wyjatek** i wygenerować na nim nowy obiekt klasy **Wyjatek** wykorzystując do tego konstruktor z argumentami: numer=1, opis=”BLAD METODY PUSH”. Następnie należy rzucić wskaźnikiem **wyjatek**.

4. W klasie **Stos** zaimplementuj metodę **int pop(int i)**, która będzie zwracała **i**-tą komórkę tablicy **tablica**. W przypadku próby przekroczenia rozmiaru tablicy (czyli dla i>9) metoda ma utworzyć wskaźnik **Wyjatek* wyjatek** i

wygenerować na nim nowy obiekt klasy **Wyjatek** wykorzystując do tego konstruktor z argumentami: numer=2, opis="BLAD METODY POP". Następnie należy rzucić wskaźnikiem **wyjatek**.

5. Utwórz klasę **Symulacja** w której znajdzie się jedno pole prywatne **Stos* stos**. W sekcji publicznej klasy utwórz trzy metody identycznie jak poniżej:

```
Symulacja()
{
cout << "Konstruktor symulacja " <<endl;
stos = new Stos();
};

~Symulacja()
{
cout << "Destruktor symulacja" << endl;
delete stos;
}

void symulacja();
```

Twoim zadaniem jest zaimplementować poza obszarem klasy **Symulacja** ciało metody **void symulacja()**.
Metoda powinna zawierać dwie pętle **for (int i=0; i<=10; i++)**, które używając klauzuli **try** próbują wywołać metody **stos->push(i, i)** oraz **stos->pop(i)**. Obie pętle zawierać powinny klauzulę **catch (Wyjatek* w)**, która łapie wskaźnik na wyjątek i wywołuje na nim metody **getNum()** oraz **getOpis()** obudowując je stosownym komunikatem tekstowym o pozyskanym numerze i opisie błędu. Zauważ, że pętle wychodzą poza granicę tablicy o rozmiarze 10 - to powinno wygenerować wyjątki w metodach push i pop.

6. Funkcja główna programu, tzn. **int main()** musi wyglądać następująco:

```
int main()
{
   Symulacja symulacja;
   symulacja.symulacja();
}
```