Politechnika Wrocławska

Techniki programowania

INP001002Wl

rok akademicki 2018/19

semestr letni

Wykład 7

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

A-1 p. 411B

Politechnika Wrocławska

Plan prezentacji

- Argumenty linii poleceń
- Szablony klas
- Wyjątki i obsługa błędów

Na podstawie:

- A. Allain, *Przewodnik dla początkujących C++*
- S. Prata, *Szkoła programowania C++*

Politechnika Wrocławska

Argumenty linii poleceń

main.cpp

```
10  //argc - liczba argumentów w linii poleceń
11  //argv - tablica wskaźników znakowych
12  int main(int argc, char *argv[])
13  {
14      //program oczekuje wywołania o jednym argumencie
15      //argv[0] - to nazwa programu
16      //argv[1] - to nazwa pliku
17      if( argc != 2 ){
18          cout << "Wywolanie: " << argv[0] << " <nazwa pliku>" << endl;
19      }
20      else{
21          //otwarcie pliku do odczytu
22          ifstream file( argv[1] );
23          //sprawdzenie czy plik został otwarty
24          if( !file.is_open() ){
25              cout << "Blad otwarcia pliku " << argv[1] << endl;
26              return -1;
27          }
28          char c;
29          //odczytywanie pliku znak po znaku
30          while( file.get(c) ){
31              cout << c;
32          }
33      }
34      return 0;
35  }
```

Politechnika Wrocławska

Szablony klas

- Biblioteka STL zawiera definicje klas szablonowych, których konkretyzacje zależą od przekazanych typów.
- Przykładowo:

  `vector<int> wekt;`

  tworzy konkretyzację szablonu klasy **`vector`** dla typu **`int`**.
- Szablony klas umożliwiają tworzenie w pełni uogólnionych deklaracji klas.

Politechnika Wrocławska

Szablony klas

- Deklaracja szablonu klasy (i funkcji) poprzedzona jest słowem kluczowym template

```
template <typename Type>
//ew. template <class Type>
```

- W definicji szablonu możemy się posługiwać słowem **Type**, tam gdzie chcemy użyć typu określonego przy tworzeniu konkretyzacji szablonu.
- Przykładowo:

```
Type items[MAX];
```

Politechnika Wrocławska

Szablony klas

myVector.h

```
1   /*Plik nagłówkowy definiujący szablon klasy myVector.*/
2   #ifndef MYVECTOR_H_INCLUDED
3   #define MYVECTOR_H_INCLUDED
4
5   //szablon klasy, który zależy od typu T
6   template <typename T>
7   class myVector{
8       public:
9           myVector();
10          myVector(int size);
11          ~myVector();
12          //deklaracja przeciążonego operatora [],
13          //który zwraca referencję do typu T
14          T& operator[](int index);
15          const T& operator[](int index) const;
16          int size() const;
17
18      private:
19          //klasa przechowuje n elementów typu T
20          //w tablicy wskazywanej przez data
21          //data jest wskaźnikiem na typ T
22          T* data;
23          int n;
24  };
```

Politechnika Wrocławska

Szablony klas

myVector.h

```
26  //szablon konstruktora wektora pustego
27  template <typename T>
28  myVector<T>::myVector():data(NULL),n(0){}
29
30  //szablon konstruktora wektora n-elementowego
31  template <typename T>
32  myVector<T>::myVector(int size):n(size){
33      data = new T[n];
34  }
35
36  //szablon destruktor
37  template <typename T>
38  myVector<T>::~myVector(){
39      delete[] data;
40  }
```

Politechnika Wrocławska

Szablony klas

myVector.h

```
42  //szablon przeciążonego operatora []
43  template <typename T>
44  T& myVector<T>::operator[](int index){
45      return data[index];
46  }
47
48  //szablon przeciążonego operatora [] dla stałego wektora
49  template <typename T>
50  const T& myVector<T>::operator[](int index) const{
51      return data[index];
52  }
53
54  //szablon metody size() - ciało funkcji nie zależy od typu T,
55  //ale zakres metody myVector<T> zależy.
56  template <typename T>
57  int myVector<T>::size() const{
58      return n;
59  }
60
61  //tak zdefiniowany szablon można rozszerzyć np. o metody pozwalające
62  //na zmienianie rozmiarów wektora
63
64  #endif // MYVECTOR_H_INCLUDED
```

Politechnika Wrocławska

Szablony klas

myVector.h | main.cpp

```
1  /*Program ilustruje wykorzystanie szablonu klasy myVector
2  zdefiniownego w pliku myVector.h*/
3  #include <iostream>
4  #include "myVector.h"
5
6  using namespace std;
7
8  int main()
9  {
10     //deklaracja wektora liczb całkowitych
11     myVector<int> vec(8);
12     //w szablonie klasy zdefiniowano metodę size()
13     for(int i=0; i<vec.size(); i++)
14         //przeciążono także operator[],
15         //co pozwala na dostęp do składowych wektora
16         vec[i] = i*i;
17     for(int i=0; i<vec.size(); i++)
18         cout << vec[i] << " ";
19     cout << endl;
20
21     //deklaracja wektora zmiennych znakowych
22     myVector<char> alphabet(26);
23     for(int i=0; i<alphabet.size(); i++)
24         alphabet[i] = 'a' + i;
25     for(int i=0; i<alphabet.size(); i++)
26         cout << alphabet[i] << " ";
27
28 }
29
```

Politechnika Wrocławska

Wyjątki i obsługa błędów

- W czasie działania programu można natrafić na problem, który powoduje zakończenie jego działania.
- Przykładowo:
  - otwieranie nieistniejącego pliku,
  - alokacja zbyt dużej ilości pamięci,
  - napotkanie nieoczekiwanej wartości.
- Do obsługi nietypowych sytuacji można wykorzystać mechanizm wyjątków.

Politechnika Wrocławska

Wyjątki i obsługa błędów

- Rozważmy przykład funkcji obliczającej średnią harmoniczną dwóch liczb

  $z = 2{,}0 \cdot x \cdot y \,/\, (x + y)$
- Dla $x = -y$ wartość funkcji jest nieokreślona.

Politechnika Wrocławska

Wyjątki i obsługa błędów

Funkcja `abort()`

- Obsługa błędu przez wywołanie funkcji **`abort()`**.

Wyjątki i obsługa błędów

Funkcja abort()

main.cpp

```
1   #include <iostream>
2   #include <cstdlib>  //abort()
3
4   using namespace std;
5
6   //deklaracja funkcji obliczającej średnią harmoniczną
7   double hmean(double a, double b);
8
9   //przykładowy program wykorzystujący funkcję hmean()
10  int main()
11  {
12      double x, y, z;
13
14      cout << "Podaj dwie liczby: ";
15      while(cin >> x >> y){
16          z = hmean(x, y);
17          cout << "Srednia harmoniczna liczb " << x << " i " << y
18               << " wynosi " << z << endl;
19          cout << "Podaj kolejna pare liczb <w, aby wyjsc>: ";
20      }
21      cout << "Koniec" << endl;
22      return 0;
23  }
```

Wyjątki i obsługa błędów

Funkcja `abort()`

main.cpp

```
25  //definicja funkcji hmean() wykorzystująca funckję abort()
26  //do obsługi sytuacji, w której a = -b.
27  double hmean(double a, double b){
28      if( a == -b ){
29          cout << "Niepoprawne argumenty funkcji hmean()" << endl;
30          abort();
31      }
32      return 2. * a * b / ( a + b );
33  }
```

Politechnika Wrocławska

Wyjątki i obsługa błędów

Funkcja zwracająca status wykonania

- Wynik działania funkcji jest zapisany w argumencie przekazanym przez wskaźnik (lub referencję), natomiast wartość zwracana informuje o statusie wykonania funkcji.

Wyjątki i obsługa błędów

Funkcja zwracająca status wykonania

main.cpp

```
1  #include <iostream>
2  #include <cfloat>   //DBL_MAX
3
4  using namespace std;
5
6  //funkcja zwraca wartość true, jeśli obliczenia
7  //przebigły poprawnie albo false w przeciwnym przypadku,
8  //natomiast wynik zostaje zapisany w trzecim argumencie wywołania
9  //przekazanym przez wskaźnik
10 bool hmean(double a, double b, double * ans);
11
12 int main()
13 {
14     double x, y, z;
15
16     cout << "Podaj dwie liczby: ";
17     while(cin >> x >> y){
18         //wywołanie funkcji zwracającej status wykonania
19         if( hmean(x, y, &z) )
20             cout << "Srednia harmoniczna liczb " << x << " i " << y
21                  << " wynosi " << z << endl;
22         else
23             cout << "Suma liczb nie moze wynosic 0 - "
24                  << "Sprobuj jeszcze raz." << endl;
25             cout << "Podaj kolejna pare liczb <w, aby wyjsc>: ";
26     }
27     cout << "Koniec" << endl;
28     return 0;
29 }
```

Politechnika Wrocławska

Wyjątki i obsługa błędów

Funkcja zwracająca status wykonania

main.cpp

```
31  bool hmean(double a, double b, double * ans){
32      if( a == -b ){
33          *ans = DBL_MAX;
34          return false;
35      }
36      else{
37          *ans = 2. * a * b / ( a + b );
38          return true;
39      }
40  }
```

Politechnika Wrocławska

Wyjątki

- Na obsługę wyjątków składają się trzy elementy:
  - zgłaszanie wyjątku,
  - umieszczenie kodu w bloku **`try`**,
  - wychwytywanie wyjątku w bloku **`catch`**.

Politechnika Wrocławska

Wyjątki

main.cpp

```
1  #include <iostream>
2
3  using namespace std;
4
5  double hmean(double a, double b);
6
7  int main()
8  {
9      double x, y, z;
10
11     cout << "Podaj dwie liczby: ";
12     while(cin >> x >> y){
13         try{
14             //wywołanie funkcji hmean() wewnątrz bloku try
15             z = hmean(x, y);
16         }
17         //w bloku catch następuje "złapanie" wyjątku
18         //i jego obsłużenie
19         catch( const char* s ){
20             cout << s << endl;
21             cout << "Podaj kolejna pare liczb: ";
22             continue;
23         }
24         cout << "Srednia harmoniczna liczb " << x << " i " << y
25              << " wynosi " << z << endl;
26         cout << "Podaj kolejna pare liczb <w, aby wyjsc>: ";
27     }
28     cout << "Koniec" << endl;
29     return 0;
30 }
```

Politechnika Wrocławska

Wyjątki

main.cpp

```
32  double hmean(double a, double b){
33      if( a == -b )
34          //zgłoszenie wyjątku w postaci ciągu znaków
35          // z wykorzystaniem instrukcji throw
36          throw "Niepoprawne argumenty funkcji hmean(): a = -b nie jest dozwolone";
37      return 2. * a * b / ( a + b );
38  }
```

Politechnika Wrocławska

Wyjątki

- Wyjątki mogą być zgłaszane w postaci obiektów.
- Wyjątki zgłaszane w różnych sytuacjach mogą być różnych typów.
- Obiekt wyjątku może zawierać dodatkowe informacje.

Politechnika Wrocławska

Wyjątki

exc_mean.h

```
1   #ifndef EXC_MEAN_H_INCLUDED
2   #define EXC_MEAN_H_INCLUDED
3
4   #include <iostream>
5
6   //definicja klasy bad_hmean - wykorzystywanej jako wyjątek
7   //zgłaszany przy obliczaniu średniej harmonicznej
8   class bad_hmean{
9       public:
10          bad_hmean(double a = 0, double b = 0) : v1(a), v2(b){}
11          void mesg();
12      private:
13          double v1;
14          double v2;
15  };
16
17  inline void bad_hmean::mesg(){
18      std::cout << "hmean(" << v1 << ", " << v2 << "): "
19                << "niepoprawne argumenty: a = -b" << std::endl;
20  }
```

Politechnika Wrocławska

Wyjątki

exc_mean.h

```
22  //definicja klasy bad_gmean - wykorzystywanej jako wyjątek
23  //zgłaszany przy obliczaniu średniej geometrycznej
24  class bad_gmean{
25      public:
26          bad_gmean(double a = 0, double b = 0) : v1(a), v2(b){}
27          const char* mesg();
28          double v1;
29          double v2;
30  };
31
32  inline const char* bad_gmean::mesg(){
33      return "Argumenty funkcji gmean() powinny byc >= 0\n";
34  }
35
36  #endif // EXC_MEAN_H_INCLUDED
```

Politechnika Wrocławska

Wyjątki

exc_mean.h | main.cpp

```
15      while(cin >> x >> y){
16          //blok try zawiera wywołania funkcji hmean() oraz gmean()
17          try{
18              z = hmean(x, y);
19              cout << "Srednia harmoniczna liczb " << x << " i " << y
20                   << " wynosi " << z << endl;
21              cout << "Srednia geometryczna liczb " << x << " i " << y
22                   << " wynosi " << gmean(x,y) << endl;
23              cout << "Podaj kolejna pare liczb <w, aby wyjsc>: ";
24          }
25          //blok catch - przechwytujący wyjątek typu bad_hmean
26          catch( bad_hmean & bh ){
27              bh.mesg();
28              cout << "Sprobuj ponownie." << endl;
29              continue;
30          }
31          //blok catch - przechwytujący wyjątek typu bad_gmean
32          catch( bad_gmean & bg ){
33              cout << bg.mesg();
34              cout << "Uzyte wartosc: " << bg.v1 << ", "
35                   << bg.v2 << endl;
36              cout << "Koniec zabawy" << endl;
37              break;
38          }
39      }
```

Politechnika Wrocławska

Wyjątki

exc_mean.h | main.cpp

```
44  double hmean(double a, double b){
45      if( a == -b )
46          throw bad_hmean(a, b);
47      return 2. * a * b / ( a + b );
48  }
49
50  double gmean(double a, double b){
51      if( a < 0 || b < 0 )
52          throw bad_gmean(a, b);
53      return sqrt( a * b );
54  }
```

Politechnika Wrocławska

Wyjątki

- Wyjątki mogą być zgłaszane nie tylko z funkcji bezpośrednio wywoływanych w bloku `try`, ale także z funkcji wywoływanych przez te funkcje.
- Do przechodzenia do właściwego miejsca obsługi wyjątku wykorzystuje się rozwijanie stosu.
- W trakcie rozwijania stosu wywoływane są automatycznie destruktory automatycznych obiektów klas utworzonych przez funkcje pośrednie.

Politechnika Wrocławska

Wyjątki

main.cpp

```
45  double hmean(double a, double b){
46      if( a == -b )
47          throw bad_hmean(a, b);
48      return 2. * a * b / ( a + b );
49  }
50
51  double gmean(double a, double b){
52      if( a < 0 || b < 0 )
53          throw bad_gmean(a, b);
54      return sqrt( a * b );
55  }
56
57  double means(double a, double b){
58      double am, hm, gm;
59      demo d2("z funkcji means()");
60      am = ( a + b ) / 2.0;
61      try{
62          hm = hmean(a, b);
63          gm = gmean(a, b);
64      }
65      catch(bad_hmean & bh){
66          bh.mesg();
67          cout << "Przechwycony w means()" << endl;
68          throw;
69      }
70      d2.show();
71      return (am + hm + gm) / 3.0;
72  }
```

Politechnika Wrocławska

Wyjątki

main.cpp

```
12  int main()
13  {
14      double x, y, z;
15      {
16          demo d1("z bloku zagniezdzonego w funkcji main()");
17
18          cout << "Podaj dwie liczby: ";
19          while(cin >> x >> y){
20              try{
21                  z = means(x, y);
22                  cout << "Srednia srednich liczb " << x << " i " << y
23                       << " wynosi " << z << endl;
24                  cout << "Podaj kolejna pare liczb: ";
25              }
26              catch( bad_hmean & bh ){
27                  bh.mesg();
28                  cout << "Sprobuj ponownie." << endl;
29                  continue;
30              }
31              catch( bad_gmean & bg ){
32                  cout << bg.mesg();
33                  cout << "Uzyte wartosc: " << bg.v1 << ", "
34                       << bg.v2 << endl;
35                  cout << "Koniec zabawy" << endl;
36                  break;
37              }
38          }
39          d1.show();
40      }
41      cout << "Koniec" << endl;
42      return 0;
43  }
```

Politechnika Wrocławska

Wyjątki

main.cpp | **demo.h** | exc_mean.h

```
1  #ifndef DEMO_H_INCLUDED
2  #define DEMO_H_INCLUDED
3  #include <string>
4
5  class demo{
6      private:
7          std::string word;
8      public:
9          demo (const char * s){
10             word = s;
11             std::cout << "Obiekt demo " << word << " utworzony." << std::endl;
12         }
13         ~demo(){
14             std::cout << "Obiekt demo " << word << " usuniety." << std::endl;
15         }
16         void show() const{
17             std::cout << "Obiekt demo " << word << " istnieje." << std::endl;
18         }
19 };
20
21 #endif // DEMO_H_INCLUDED
```

Politechnika Wrocławska

Wyjątki

- Biblioteka standardowa zawiera definicję klasy **`exception`**.
- Własne wyjątki można tworzyć dziedzicząc po klasie **`exception`**.

Politechnika Wrocławska

Wyjątki

main.cpp | exc_mean.h

```
1  #ifndef EXC_MEAN_H_INCLUDED
2  #define EXC_MEAN_H_INCLUDED
3
4  #include <exception>
5
6  class bad_hmean : public std::exception {
7      public:
8          virtual char const * what() const throw() {return "bla";}
9  };
10
11 class bad_gmean : public std::exception {
12     public:
13         virtual char const * what() const throw() {return "bla2";}
14 };
15
16
17
18 #endif // EXC_MEAN_H_INCLUDED
```

Politechnika Wrocławska

Podsumowanie

- Argumenty linii poleceń
- Szablony klas
- Wyjątki i obsługa błędów