Politechnika Wrocławska

Techniki programowania

INP001002Wl

rok akademicki 2018/19

semestr letni

Wykład 5

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

A-1 p. 411B

Politechnika Wrocławska

Plan prezentacji

- Przestrzenie nazw
- Standardowa biblioteka szablonów (*Standard Template Library STL*)
- Operacje plikowego wejścia/wyjścia

Na podstawie:

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

Politechnika Wrocławska

Przestrzenie nazw

- Nazwy w języku C++ mogą odnosić się do zmiennych, funkcji, struktur, wyliczeń, klas, składowych struktur i klas
- Dla rosnących projektów rośnie ryzyko kolizji tych nazw
- Dodatkowo ryzyko zwiększa się w projektach wykorzystujących biblioteki zewnętrzne.
- W celu uniknięcia konfliktów wprowadzono mechanizm przestrzeni nazw.

Politechnika Wrocławska

Przestrzenie nazw

- Polecenie

`using namespace` ***`nazwa_przestrzeni;`***

udostępnia wszystkie nazwy z przestrzeni ***`nazwa_przestrzeni`***

- Przykładowo

`using namespace std;`

udostępnia wszystkie nazwy z przestrzeni **`std`**.

Politechnika Wrocławska

Przestrzenie nazw

- Polecenie

```
using namespace std;
```

udostępnia wszystkie nazwy z przestrzeni **`std`**.

- Dzięki temu możemy korzystać bezpośrednio z obiektów **`cout`**, **`cin`**.
- Nie musimy odwoływać się do pełnych nazw:

`std::cout`, **`std::cin`**.

Przestrzenie nazw

Definiowanie przestrzeni nazw

- Przykład

```
namespace cprogramming{
      int x;
}//brak średnika
```

- Dostęp do zmiennej x można uzyskać pisząc
  `cprogramming::x`,
  dołączając całą przestrzeń nazw
  `using namespace cprogramming;`
  lub dołączając tylko wybraną nazwę
  z przestrzeni
  `using cprogramming::x;`

Przestrzenie nazw

Zagnieżdżanie przestrzeni nazw

- Przykład

```
namespace com{
namespace cprogramming
{
    int x;
}}
```

Politechnika Wrocławska

Przestrzenie nazw

- Przykład

do istniejącej przestrzeni nazw

```
namespace com{
      int x;
}
```

można dodać nowy element

```
...
namespace com{
      double y;
}
```

Politechnika Wrocławska

Przestrzenie nazw

Kiedy stosować instrukcję `using namespace`

- Zazwyczaj nie należy umieszczać deklaracji **`using`** w plikach nagłówkowych, a tylko w plikach źródłowych.
- Zalecenie:
  - w plikach nagłówkowych stosuj pełne nazwy,
  - w plikach źródłowych można wykorzystywać deklarację **`using`**.

Politechnika Wrocławska

Standardowa biblioteka szablonów

- Standardowa biblioteka szablonów udostępnia zestaw szablonów reprezentujących kontenery, iteratory, obiekty funkcyjne oraz algorytmy.
- Kontener to jednostka, która pozwala przechowywać grupę wartości ustalonego typu – podobnie jak tablica.

Politechnika Wrocławska

Standardowa biblioteka szablonów

- Wykorzystanie standardowej biblioteki szablonów może wykluczyć potrzebę tworzenia niestandardowych struktur danych.
- Aby korzystać z biblioteki STL należy poznać interfejsy szablonów.

Politechnika Wrocławska

Wektor

- Wektor dostępny w bibliotece STL przypomina tablicę z dynamicznym zarządzaniem pamięcią.

Politechnika Wrocławska

Wektor

- Deklaracja wektora

main.cpp

```
1   #include <iostream>
2   #include <vector>
3
4   using namespace std;
5
6   int main()
7   {
8       //deklaracja 4-elementowej tablicy liczb całkowitych
9       int table[4];
10
11      //deklaracja 4-elementowego wektora liczb całkowitych
12      vector<int> vec(4);
```

Politechnika Wrocławska

Wektor

- W wektorze można przechowywać dane dowolnego typu.
- Deklarując wektor należy określić typ danych.

  `vector<`*`typ`*`>`
- Wektor znajduje się w przestrzeni nazw **`std`**.

Politechnika Wrocławska

Wektor

- Szablon wektora udostępnia wiele różnych metod, np.:
  - **`at`** – zwraca wartość na podanej pozycji,
  - **`insert`** – wstawia podaną wartość na podanej pozycji,
  - **`push_back`** – dopisuje podaną wartość na końcu wektora,
  - **`pop_back`** – usuwa ostatni element wektora,
  - **`shrink_to_fit`** - dostosowuje zaalokowaną pamięć do wypełnienia wektora,
  - **`clear`** – usuwa wszystkie elementy wektora,
  - **`size`** – zwraca liczbę elementów w wektorze,
  - **`capacity`** – zwraca liczbę elementów, które można przechowywać w wektorze bez realokacji pamięci

Politechnika Wrocławska

Wektor

- Dla wektorów przeciążono także operator[], dzięki czemu do elementów wektora można się odwoływać jak do elementów tablicy

main.cpp

```
14      //zerowanie wszystkich elementów tablicy i wektora
15      for(int i = 0; i<4; i++){
16          table[i] = 0;
17          vec[i]   = 0;
18      }
19
20      //wyświetlenie rozmiaru wektora
21      cout << "rozmiar wektora: " << vec.size() << endl;
22
23      //dodanie elementu do wektora (dopisanie na końcu)
24      vec.push_back(17);
```

Politechnika Wrocławska

Mapa

- Struktura danych mapa z biblioteki STL pozwala przechowywać wartości w układzie: klucz/wartość.
- Klucz i wartość mogą być dowolnych typów.
- Mapa znajduje się w przestrzeni nazw **`std`**.
- Deklaracja mapy:

```
map<typ_klucza, typ_wartosci> mapa;
```

Politechnika Wrocławska

Mapa

- Przykładowa deklaracja mapy:

main.cpp

```
1  #include <iostream>
2  #include <map>
3  #include <string>
4
5  using namespace std;
6
7  void wyswietlMape(map<string, string>);
8
9  int main()
10 {
11     //deklaracja zmiennej typu mapa
12     //klucz   typu string
13     //wartość typu string
14     map<string, string> nazwa_email;
```

Politechnika Wrocławska

Mapa

- Dodanie wartości do mapy:

main.cpp

```
16      //dodanie do mapy nowego elementu
17      //adresu e-mail: "alex.allain@cprogramming.com"
18      //z kluczem:     "Alex Allain"
19      nazwa_email["Alex Allain"] = "alex.allain@cprogramming.com";
```

- Dostęp do elementu w mapie:

main.cpp

```
21      //wyświetlenie wartości powiązanej z kluczem "Alex Allain"
22      cout << nazwa_email["Alex Allain"] << endl;
```

Politechnika Wrocławska

Mapa

- Szablon mapy udostępnia wiele różnych metod, np.:
  - **`at`** – zwraca wartość dla podanego klucza,
  - **`erase`** – usuwa wartość o podanym kluczu,
  - **`clear`** – usuwa wszystkie elementy mapy,
  - **`size`** – zwraca liczbę elementów w mapie,
  - **`empty`** – zwraca informację, czy mapa jest pusta.

Politechnika Wrocławska

Iteratory

- Do przejścia po wszystkich elementach mapy można wykorzystać iterator.
- Deklaracja iteratora dla mapy z przykładu:

```
map<string, string>::iterator itr;
```

Politechnika Wrocławska

Iteratory

- Również szablon wektor udostępnia iterator.
- Wykorzystując metody begin() oraz end() możemy iteracyjnie przejść przez wszystkie elementy wektora

main.cpp

```
26      //wyświetlenie elementów wektora z wykorzystaniem iteratora
27      for( vector<int>::iterator itr = vec.begin();
28           itr != vec.end();
29           ++itr ){
30          cout << *itr << endl;
31      }
32      cout << endl;
```

Politechnika Wrocławska

Iteratory

- O iteratorze można myśleć jako o ulepszonym wskaźniku.
- Dla iteratorów zaimplementowano operatory inkrementacji **++** oraz dereferencji *****.

main.cpp

```
26      //wyświetlenie elementów wektora z wykorzystaniem iteratora
27      for( vector<int>::iterator itr = vec.begin();
28           itr != vec.end();
29           ++itr ){
30          cout << *itr << endl;
31      }
32      cout << endl;
```

Politechnika Wrocławska

Iteratory

- Zmodyfikowana pętla z poprzedniego przykładu - dzięki zapamiętaniu iteratora dla końca wektora (**`koniec`**) nie trzeba wywoływać metody **`end()`**, przy sprawdzaniu warunku kontynuacji pętli.

main.cpp

```
34      //wyświetlenie elementów wektora z wykorzystaniem iteratora
35      //bez wywoływania metody end() przy każdym przebiegu pętli
36      for( vector<int>::iterator itr = vec.begin(), koniec = vec.end();
37           itr != koniec;
38           ++itr ){
39          cout << *itr << endl;
40      }
41      cout << endl;
```

Politechnika Wrocławska

Iteratory

- Podobnie można wyświetlić zawartość mapy:

main.cpp

```
38  //funkcja wyswietlMape() wykorzystuje iterator
39  //aby przejść przez wyszystkie elementy mapy
40  //i je wyświetlić
41  void wyswietlMape(map<string, string> mapa){
42      for( map<string, string>::iterator itr = mapa.begin(), koniec = mapa.end();
43           itr != koniec; ++itr ){
44          cout << itr->first << " --> " << itr->second << endl;
45      }
46  }
```

Politechnika Wrocławska

Iteratory

- Dostęp do klucza i wartości uzyskuje się przez pola **`first`** i **`second`**

main.cpp

```
38  //funkcja wyswietlMape() wykorzystuje iterator
39  //aby przejść przez wyszystkie elementy mapy
40  //i je wyświetlić
41  void wyswietlMape(map<string, string> mapa){
42      for( map<string, string>::iterator itr = mapa.begin(), koniec = mapa.end();
43           itr != koniec; ++itr ){
44          cout << itr->first << " --> " << itr->second << endl;
45      }
46  }
```

Politechnika Wrocławska

Iteratory

- Wykorzystanie metody **`find()`** do wyszukania w mapie elementu o podanym kluczu

main.cpp

```
27      //wykorzystanie metody find(), bo znalezienia zadanego elementu mapy
28      map<string, string>::iterator itr = nazwa_email.find("Alex Allain");
29      if( itr != nazwa_email.end() ){
30          cout << "To adres Alexa: " << itr->second;
31      }
```

Politechnika Wrocławska

Iteratory

- Poprawiona funkcja wyswieltMape() pobiera mapę jako referencję zadeklarowaną jako **`const`**

main.cpp

```
7   //Funkcja wyswietlMape pobiera mapę przez referencję.
8   //Dodatkowo zadeklarowano referencję jako const,
9   //bo funkcja tylko wyświetla mapę; nie modyfikuje jej.
10  void wyswietlMape(const map<string, string>&);
```

- W funkcji wykorzystywany jest iterator **`const_iterator`**

main.cpp

```
33  //Ponieważ argument mapa zadeklarowany jest jako const,
34  //to do obsługi mapy należy wykorzystać iterator,
35  //który nie pozwoli na zmianę kluczy/wartości mapy.
36  //Jest to const_iterator.
37  void wyswietlMape(const map<string, string>& mapa){
38      for( map<string, string>::const_iterator itr = mapa.begin(), koniec = mapa.end();
39           itr != koniec; ++itr ){
40          cout << itr->first << " --> " << itr->second << endl;
41      }
42  }
```

Politechnika Wrocławska

Dostęp do plików

- Odczyt i zapis plików przypomina korzystanie z obiektów **`cin`** oraz **`cout`**.
- Dwie klasy **`ifstream`** oraz **`ofstream`** pozwalają obsługiwać strumienie wejściowy oraz wyjściowy.

Politechnika Wrocławska

Otwieranie pliku do odczytu

- Konstruktor klasy **`ifstream`** pobiera jako argument nazwę pliku.
- Nazwa pliku może być podana z pełną ścieżką dostępu.
- Jeżeli nie będzie ścieżki dostępu to plik będzie poszukiwany w katalogu roboczym.

Politechnika Wrocławska

Otwieranie pliku do odczytu

- Przykładowe wywołanie konstruktora:

Start here | ifstream.cpp

```
1  #include <fstream> //plik nagłówkowy
2
3  using namespace std;
4
5  int main(){
6      //wywołanie konstruktora klasy ifstream
7      ifstream read_file("myfile.txt");
8      return 0;
9  }
10
```

Politechnika Wrocławska

Otwieranie pliku do odczytu

- Pracując z plikami należy sprawdzać, czy przebieg programu jest zgodny z oczekiwaniami.
- Do sprawdzenia, czy plik jest prawidłowo otwarty służy metoda `is_open()`.

Politechnika Wrocławska

Otwieranie pliku do odczytu

Start here | ifstream_check_file_status.cpp

```
1   #include <fstream>
2   #include <iostream>
3
4   using namespace std;
5
6   int main(){
7       ifstream read_file("myfile.txt");
8       //metoda is_open() pozwala sprawdzić,
9       //czy udało się otworzyć plik
10      if( !read_file.is_open() )
11          cout << "Blad otwarcia pliku!" << endl;
12      return 0;
13  }
14
```

Politechnika Wrocławska

Odczyt pliku

- Jeżeli plik jest otwarty to z obiektu klasy **`ifstream`** można korzystać jak z obiektu **`cin`**.

Politechnika Wrocławska

Odczyt pliku

Start here | ifstream_read_file.cpp

```cpp
#include <fstream>
#include <iostream>

using namespace std;

int main(){
    ifstream read_file("myfile.txt");
    //jeśli nie udało się otworzyć pliku kończymy
    //działanie programu
    if( !read_file.is_open() ){
        cout << "Blad otwarcia pliku!" << endl;
        return -1;
    }

    //w przeciwnym przypadku wczytujemy z pliku liczbę
    int number;
    read_file >> number;

    return 0;
}
```

Politechnika Wrocławska

Odczyt pliku

- Pracując z plikami należy sprawdzać, czy przebieg programu jest zgodny z oczekiwaniami.
- Wartość zwracana przez operator >> można wykorzystać do sprawdzenia, czy dane są prawidłowo wczytane.

Politechnika Wrocławska

Odczyt pliku

Start here | ifstream_check_readout_status.cpp

```cpp
#include <fstream>
#include <iostream>

using namespace std;

int main(){
    ifstream read_file("myfile.txt");
    if( !read_file.is_open() ){
        cout << "Blad otwarcia pliku!" << endl;
        return -1;
    }

    int number;
    //sprawdzenie poprawnego odczytu danych
    if( read_file >> number )
        cout << "Wczytana wartosc to: " << number;

    return 0;
}
```

Politechnika Wrocławska

Odczyt pliku

- Przykład odczytu pliku - czytany plik zawiera 10-elementową listę z wynikami

Start here | results.cpp

```
1  #include <fstream>
2  #include <iostream>
3  #include <vector>
4
5  using namespace std;
6
7  int main(){
8      ifstream read_file("results.txt");
9      if( !read_file.is_open() ){
10         cout << "Blad otwarcia pliku!" << endl;
11         return -1;
12     }
13
14     vector<int> results;
15     for(int i=0; i<10; i++){
16         int result;
17         read_file >> result;
18         results.push_back(result);
19     }
20
21     return 0;
22 }
23
```

Odczyt pliku - koniec pliku

- Przykład odczytu pliku - przerywamy jeśli nie udało się wczytać kolejnej liczby

Start here × results_eof.cpp ×

```
14      vector<int> results;
15      for(int i=0; i<10; i++){
16          int result;
17          //przerwanie nastąpi jeśli wyników
18          //jest mniej niż 10
19          if( ! ( read_file >> result ) ){
20              break;
21          }
22          results.push_back(result);
23      }
```

Politechnika Wrocławska

Odczyt pliku

- Do sprawdzenia, czy w trakcie odczytu wystąpił błąd można wykorzystać metodę **`fail()`**.
- Do sprawdzenia, czy w trakcie odczytu osiągnięto koniec pliku można wykorzystać metodę **`eof()`**.
- Po osięgnięciu końca pliku należy użyć metody **`clear()`**, aby przeprowadzać inne operacje plikowe.

Politechnika Wrocławska

Zapis pliku

- Do zapisu pliku służy klasa **`ofstream`**. Obiektu klasy **`ofstream`** używamy jak obiektu **`cout`**.

Politechnika Wrocławska

Zapis pliku

Start here | ofstream.cpp

```cpp
#include <fstream>
#include <iostream>
#include <vector>

using namespace std;

int main(){
    //otwarcie pliku do zapisu
    ofstream write_file("results2.txt");

    if( !write_file.is_open() ){
        cout << "Blad otwarcia pliku!" << endl;
        return -1;
    }

    //wypisanie do pliku liczb od 10 do 1
    for(int i=10; i>0; i--){
        write_file << i << " ";
    }

    return 0;
}
```

Politechnika Wrocławska

Zapis pliku

- Konstruktor klasy **`ofstream`** przyjmuje jako drugi argument określający sposób obsługi pliku

`ios::app`	dołączanie do pliku
`ios::ate`	ustawienie bieżącej pozycji na końcu
`ios::trunc`	usunięcie całej zawartości pliku
`ios::out`	umożliwia zapis
`ios::binary`	umożliwia operacje binarne

- Przykładowo:

```
ofstream file("nazwa", ios::app | ios::binary )
```

Politechnika Wrocławska

Pozycja w pliku

- Do odczytania pozycji w pliku służą funkcje **`tellg`** oraz **`tellp`**.
- Do ustawienia pozycji w pliku służą funkcje **`seekg`** oraz **`seekp`**.

Politechnika Wrocławska

Pozycja w pliku

- Trzy opcje dotyczące zmiany pozycji w pliku:

`ios_base::beg`	początek
`ios_base::cur`	pozycja bieżąca
`ios_base::end`	koniec

Politechnika Wrocławska

Pozycja w pliku

- Przykładowy program ilustrujący odczyt i ustawienie pozycji w pliku.
- Odczyt i zapis następuje do jednego pliku, zatem korzystamy z klasy **`fstream`**.

Politechnika Wrocławska

Pozycja w pliku

Start here × **streampos.cpp** ×

```
1  #include<fstream>
2  #include<iostream>
3  #include<vector>
4
5  using namespace std;
6
7  int main(){
8      //otwarcie pliku do zapisu/odczytu
9      fstream file("results2.txt", ios::in | ios::out );
10     if( !file.is_open() ){
11         cout << "Blad otwarcia pliku!" << endl;
12         return -1;
13     }
14
15     int new_result;
16     cout << "Podaj nowy wynik: ";
17     cin  >> new_result;
18
```

Politechnika Wrocławska

Pozycja w pliku

Start here | streampos.cpp

```
19      //zapamiętujemy pozycję w pliku przed aktualnym wynikiem
20      streampos prev_result_pos = file.tellg();
21      int current_result;
22      //wczytujemy kolejne wyniki
23      while( file >> current_result ){
24          //dopóki nie trafimy na mniejszy, niż nowy wynik
25          if( current_result < new_result ){
26              break;
27          }
28          prev_result_pos = file.tellg();
29      }
30
31      //sprawdzamy, czy przerwanie pętli nastąpiło z powodu
32      //błędu odczytu lub końca pliku
33      if( file.fail() && !file.eof() ){
34          cout << "Blad odczytu pliku!";
35          return -2;
36      }
37
38      //czyścimy status błedów
39      file.clear();
40
```

Politechnika Wrocławska

Pozycja w pliku

Start here | streampos.cpp

```
41      //ustawiamy pozycję odczytu w pliku
42      //na zapamiętaną pozycję
43      //przed pierwszym wynikiem mniejszym niż nowy
44      file.seekg( prev_result_pos );
45
46      //wczytujemy kolejne wyniki do wektora
47      vector<int> results;
48      while( file >> current_result ){
49          results.push_back(current_result);
50      }
51
52      //upewniamy się, że dotarliśmy do końca pliku
53      if( !file.eof() ){
54          cout << "Blad odczytu pliku!";
55          return 0;
56      }
57
58      //czyścimy status błedów
59      file.clear();
60
```

Politechnika Wrocławska

Pozycja w pliku

Start here | streampos.cpp

```
61      //ponownie ustawiamy pozycję odczytu w pliku
62      //na zapamiętaną pozycję
63      file.seekp(prev_result_pos);
64
65      //jeśli nie jesteśmy na początku pliku to dostawiamy spację
66      if( prev_result_pos != std::streampos(0) ){
67          file << " ";
68      }
69      //zapisujemy nowy wynik
70      file << new_result << " ";
71
72      //a następnie wypisujemy zawartość wektora
73      for(vector<int>::iterator itr = results.begin(), end = results.end();
74          itr != end;
75          ++itr){
76          file << *itr << " ";
77      }
78  }
```

Politechnika Wrocławska

Pliki binarne

- Wykorzystanie plików binarnych pozwala znacząco zmniejszyć ich objętość (względem plików tekstowych).

Politechnika Wrocławska

Pliki binarne

- Otwarcie pliku w trybie binarnym:

```
ofstream file( "data.bin", ios::binary );
```

- Rzutowanie na typ **`char*`** umożliwia zapis bajt po bajcie

- Metoda **`write`** zapisuje dane (traktowane jako tablica **`char`**) o podanym rozmiarze (w bajtach)

Politechnika Wrocławska

Pliki binarne

main.cpp

```
1  #include <fstream>
2  #include <string>
3  #include <iostream>
4
5  using namespace std;
6
7  struct Player{
8      int age;
9      int score;
10     string name;
11 };
12
13 int main()
14 {
15     ///deklaracja i inicjalizacja struktury danych
16     Player player, player2;
17     player.age = 19;
18     player.score = 200;
19     player.name = "John";
20
```

Politechnika Wrocławska

Pliki binarne

main.cpp

```
21      ///zapis do pliku
22      //otworzenie pliku do odczytu/zapisu w trybie binarnym z nadpisaniem pliku
23      fstream file( "data.bin", ios::trunc | ios::binary | ios::in | ios:: out );
24
25      //zapis pola age
26      file.write( reinterpret_cast<char*>(&player.age),    sizeof(player.age) );
27
28      //zapis pola score
29      file.write( reinterpret_cast<char*>(&player.score), sizeof(player.score) );
30
31      //zapis pola name
32      //zapis długości łańcucha ...
33      int length = player.name.length();
34      file.write( reinterpret_cast<char*>(&length),    sizeof(length) );
35      //... oraz jego zawartości
36      file.write( player.name.c_str(), player.name.length() + 1 );
37
```

Politechnika Wrocławska

Pliki binarne

main.cpp

```
38      ///odczytanie danych z pliku
39      //ustawienie pozycji z pliku na początek
40      file.seekg( 0, ios::beg );
41      //odczyt sizeof(player2.age) bajtów z pliku bajt po bajcie pod adres &player2.age
42      if( !file.read( reinterpret_cast<char*>(&player2.age),   sizeof(player2.age) ) ){
43          cout << "Blad otwarcia pliku" << endl;
44          return -1;
45      }
46      //odczyt wyniku
47      if( !file.read( reinterpret_cast<char*>(&player2.score), sizeof(player2.score) ) ){
48          cout << "Blad otwarcia pliku" << endl;
49          return -1;
50      }
51
```

Politechnika Wrocławska

Pliki binarne

main.cpp

```
52      //odczyt nazwy
53      int length2;
54      //długość łańucha
55      if( !file.read( reinterpret_cast<char*>(&length2) , sizeof(length2) ) ){
56          cout << "Blad odczytu z pliku" << endl;
57          return -2;
58      }
59
60      //i sam łańcuch
61      if( length2 > 0 && length2 < 10000 ){
62          //wykorzystując pomocniczą tablicę
63          char* string_buf = new char[ length2+1 ];
64          if( !file.read( string_buf, length2 + 1 ) ){
65              delete[] string_buf;
66              cout << "Blad odczytu z pliku" << endl;
67              return -1;
68          }
69          if( string_buf[length2] == 0 ){
70              player2.name = string(string_buf);
71          }
72          delete[] string_buf;
73      }
74
75      cout << player2.age << " " << player2.score << " " << player2.name << endl;
76
77      return 0;
78  }
```

Politechnika Wrocławska

Podsumowanie

- Przestrzenie nazw
- Standardowa biblioteka szablonów (*Standard Template Library STL*)
- Operacje zapisu/odczytu plików