Początki z C++ 3

3.1 Klasa string

Klasa **string**, umieszczona jest w przestrzeni nazw **std::**. Nazwa **string** stanowi typ zmiennej. Deklaracja zmiennej wygląda więc następująco:

```
std::string nazwa_zmiennej;
```

Wczytywanie i zapisywanie tekstu za pomocą cin i cout odbywa się dokładnie tak samo, jak w przypadku zwykłej zmiennej. **Klasa string** dynamicznie zarządza danymi i alokuje bądź zwalnia sobie pamięć w zależności od potrzeb.

3.2 Przydatne funkcje

3.2.1 Wczytywanie wiersza znaków - .getline()

Żeby pobrać wiersz znaków, możemy skorzystać z funkcji getline() zawartej w klasie string. Funkcja ta przyjmuje trzy parametry - **strumień**, z którego będą wczytywane dane, **nazwę zmiennej**, do której zostanie wpisany wiersz znaków oraz **trzeci, opcjonalny parametr**, który informuje nas, po jakim znaku wczytywanie wiersza ma zostać przerwane. Domyślnie jest to “\\n” czyli znak końca wiersza - enter.

Przykładowe zastosowanie:

```
#include <iostream>
#include <string>

int main()
{
	std::string wyraz;
	getline(std::cin, wyraz);
	std::cout << "Zawartosc zmiennej wyraz: " << wyraz << endl;
	system("pause");
	return 0;
}
```

Po wpisaniu dowolnego wiersza znaków i przyciśnięcia klawisza enter, wiersz znaków zostanie wpisany do zmiennej wyraz i wyświetlony na ekranie.

3.2.2 .length()

Aby sprawdzić długość tekstu jaki znajduje się w zmiennej **string**, musimy wywołać funkcję, która znajduje się wewnątrz **klasy string**. Zadanie to może brzmieć strasznie, ale jak zobaczysz jest to bardzo prosta i wygodna sprawa. Oto przykład:

string wyraz;
getline(std::cin, wyraz);
cout << wyraz.length();

3.3 Łączenie (konkatenacja) tekstów

Jeśli mamy dwie lub więcej zmiennych typu **string** i chcemy je połączyć tak, aby dostać zmienną w której będziemy mieli wszystkie teksty połączone razem wystarczy wykorzystać do tego operator matematyczny **+**. Poniższy przykład demonstruje tą własność:

```
string wyraz, nastepny, wynik;
cout << "Podaj tekst: ";
getline( cin, wyraz );
cout << "Podaj drugi tekst: ";
getline( cin, nastepny );
wynik = wyraz + " wlasny napis " + nastepny;
```

3.4 Porównanie dwóch zmiennych typu string.

Dwa „stringi” mogą być porównane z użyciem operatorów **==**, **!=**, **<**, **<=**, **>** i **>=**: Wszystkie te operatory używają bazowej metody `std::string::compare()` do wykonania porównania I zwracania wartości logicznej. Działanie tych funkcji może być interpretowane w następujący sposób:

```
std::string str1 = "Foo";
std::string str2 = "Bar";
str1 < str2
str1 > str2
str1 <= str2
str1 >= str2
str1 == str2
str1 != str2
```

Operator ==:

Jeśli str1.length() jest równe str2.length() i każda para znaków jest sobie równa zwraca true, w innym wypadku zwracana wartość to false.

Operator !=:

Jeśli str1.length() nie jest równe str2.length() i co najmniej jedna para znaków jest różna zwraca true, w innym wypadku zwracana wartość to false.

Operator < i >:

Operator porównuje pierwszą różną parę pod względem jej wartości ASCII i zwraca odpowiednią wartość logiczną.

Operator <= oraz >=:

Operator porównuje pierwszą różną parę pod względem jej wartości ASCII i zwraca odpowiednią wartość logiczną. Dodatkowo zwraca true, gdy każda para będzie równa.

Uwaga! Termin para znaków oznacza odpowiednie znaki w obu łańcuchach na tych samych pozycjach. Dla lepszego zrozumienia, jeśli dwa przykładowe ciągi to str1 i str2, a ich długości to odpowiednio n i m pary znaków obu łańcuchów oznaczają pary str1 [i] i str2 [i], gdzie i = 0, 1, 2, ..., max (n, m). Jeśli dla jakiegokolwiek i gdzie odpowiedni znak nie istnieje, to znaczy, gdy i jest większy lub równy n lub m, byłby uważana za najniższą wartość.

Przykład:

```
std::string str1 = "Barr";
std::string str2 = "Bar";
std::cout << (str2 < str1);
```

Kroki są następujące:

1. Porównanie pierwszej pary, ‘B’ == ‘B’ – dalej.
2. Porównanie drugiej pary, ‘a’ == ‘a’ – dalej.
3. Porównanie trzeciej pary, ‘r’ == ‘r’ – dalej.
4. Zakres zmiennej str2 został wyczerpany, a zmienna str1 ma wciąż jeden znak, stąd wiadomo, że str2 < str1.

3.5 Podział zmiennej typu string

Aby podzielić string'a używamy funkcji `std::string::substr()`, są dwie możliwości użycia tej funkcji:

Pierwsza z nich przyjmuje jeden parametr, który określa pozycję, od której chcemy zacząć. Powinien znajdować się w zakresie od zera do str.length().

```
std::string str = "Witajcie na zajeciach!";
std::string newstr = str.substr(9); // "na zajeciach!"
```

Druga opcja przyjmuje dwa parametry. Pozycję początkową i całkowitą długość nowego substringa. Bez względu na długość, podłańcuch nigdy nie przekroczy końca ciągu źródłowego.

```
std::string str = "Witajcie na zajeciach!";
std::string newstr = str.substr(9,2); // "na"
```

3.6 "Zamiana" tekstu

Jeśli chcesz zamienić część stringa, możesz użyć metody **.replace().**
Posiada ona wiele przydatnych przeładowań:

```
//Definicja zmiennych string
std::string pierwszy = "Hello foo, bar and world!";
std::string drugi = "Hello foobar";

//1)
pierwszy.replace(6, 3, "bar"); //wynik = "Hello bar, bar and world!"

//2)
pierwszy.replace(pierwszy.begin() + 6, pierwszy.end(), "nobody!");
//"Hello nobody!"

//3)
pierwszy.replace(19, 5, drugi, 6, 6); //"Hello foo, bar and foobar!" -
po pierwszych 19 znakach stringu pierwszy, zamień następne 5 znaków na
string drugi (po 6 znakach weź 6 kolejnych)
```

Tylko w wersji >= C++ 14

```
//4)
pierwszy.replace(19, 5, drugi, 6); //"Hello foo, bar and foobar!" -to
samo co 3), tylko że ze stringa drugi pobierane są wszystkie znaki
stringa po sześciu pierwszych

//5)
pierwszy.replace(pierwszy.begin(), pierwszy.begin() + 5,
pierwszy.begin() + 6, pierwszy.begin() + 9);
//"foo foo, bar and world!"

//6)
pierwszy.replace(0, 5, 3, 'z'); //"zzz foo, bar and world!"

//7)
pierwszy.replace(pierwszy.begin() + 6, pierwszy.begin() + 9, 3, 'x');
//"Hello xxx, bar and world!"
```

Tylko w wersji >= C++ 11

```
pierwszy.replace(pierwszy.begin(), pierwszy.begin() + 5, { 'x', 'y',
'z'});
//"xyz foo, bar and world!"
```

3.7 Pętla przez każdy element stringa.

`std::string` obsługuje iteratory, więc możesz używać pętli opartej na odległościach do iterowania każdego znaku:

Tylko w wersji >= C++ 11

```
std::string str = "Witaj swiecie!";
for (auto c : str)
    std::cout << c;
```

Można też użyć “tradycyjnej” pętli:

```
std::string str = "Witaj swiecie!";
for (std::size_t i = 0; i < str.length(); ++i)
    std::cout << str[i];
```

3.8 Dostęp do znaku

Istnieje kilka sposobów wyodrębniania znaków ze std :: string, które nieco różnią się od siebie.

```
std::string str("Witaj swiecie!");
```

operator[](n)
Zwraca referencję do znaku w indeksie n.

std::string::operator[] nie jest sprawdzany pod względem granic i nie generuje wyjątku. Wywołanie jest odpowiedzialne za zapewnienie, że indeks znajduje się w zakresie ciągu znaków:

```
char c = lancuch[5]; //"j"
```

at(n)
Zwraca znak znajdujący się na miejscu o indeksie (n).
Gdy podamy indeks większy niż długość stringa zostanie zwrócony wyjątek std::out_of_range.

```
std::string str = "Witaj swiecie!";
char c = str.at(7); // 'w'
```

Oba te przykłady spowodują niezdefiniowane zachowanie, jeśli ciąg znaków jest pusty

front()

Zwraca pierwszy znak danego stringa.

```
std::string str = "Witaj swiecie!";
char c = str.front() // 'W'
```

back()
Zwraca referencję do ostatniego znaku:

```
char c = str.back(); // '!'
```

```
    Nie bój się wielkiego kroku. Nie pokonasz przepaści dwoma małymi.
                          ~David Loyd George
```