Języki i paradygmaty programowania 2

Laboratorium 5. Dynamiczny przydział pamięci dla klas.

1. Utwórz klasę MyString zawierającą char*str, int len – długość ciągu. Przygotuj konstruktor domyślny, sparametryzowany(parametr char *s) oraz destruktor. Przygotuj nagłówek klasy oraz implementację w dwóch plikach.

   Zwróć uwagę na konstruktor sparametryzowany. Powinien on wyglądać mniej więcej tak:

```
MyString::MyString(const char* s)
{
        len = strlen(s); //sprawdzamy długość ciągu z konstruktora
        str = new char[len +1]; //alokujemy niezbędną ilość pamięci dla składowej
klasy
        strcpy_s(str, len, s); //kopiujemy tablice z konstruktora do składowej klasy
}
```

   Przygotuj podobny konstruktor domyślny, który alokuje tablicę na 4 znaki i będzie przechowywać słowo „test”.

   Co się stanie jeżeli zamiast strcpy napiszesz:
   str =s;

   Przygotuj destruktor zwalniający pamięć. Dodatkowo umieść w nim komendę:
   cout << "destruktor" << endl;

   Zaimplementuj przeciążony operator << jako funkcję zaprzyjaźnioną.

2. Przygotuj teraz dwie funkcje po za klasą MyString.

```
void test1(MyString &s)
{
        cout << "Referencja" << endl;
        cout << s << endl;
 }

void test2(MyString s)
{
        cout << "Wartosc" << endl;
        cout << s << endl;
}
```

   Przetestuj ich działanie. Jak wywoływane są destruktory?

3. Przetestuj poniższy fragment kodu:

```
MyString a((char*)"aaaaaa");
MyString b = a; // Jaki konstruktor jest tutaj wywoływany?
```

4. Specjalne metody klasy.

Problemy z wykonaniem zadań 2 oraz 3 wynikają z faktu, że pewne metody klasy generowane są automatycznie. Są to:

a) Konstruktor domyślny
b) Operator przypisania
c) Konstruktor kopiujący
d) Domyślny destruktor

Generalnie w C++11 zdefiniowano jeszcze 2 takie metody: konstruktor przenoszący oraz przenoszący operator przypisania. Nie będą one treścią laboratorium jednak zachęcam do zapoznania się z materiałem na ich temat https://devcode.pl/cpp11-semantyka-przeniesienia/#Konstruktor_przenoszacy_oraz_przenoszacy_operator_przypisania

Konstruktor kopiujący

Wykorzystywany jest do kopiowania obiektu do nowo utworzonego obiektu tej samej klasy. Jest więc stosowany wyłącznie podczas inicjalizacji (przekazanie obiektu do funkcji przez wartość).

Budowa konstruktora:

MyString(const MyString&)

Przygotujmy jego najprostszą implementację

```
MyString::MyString(const MyString&)
{
	cout << "Jestem konstruktorem kopiującym" << endl;
}
```

Następnie przetestujmy następujący fragment kodu:

```
MyString a("aaaaaa");
MyString b = a;
MyString c(a);
MyString *d =new MyString(a);
test1(a);
test2(a);
```

W których przypadkach wywoływany jest konstruktor kopii?

Domyślny konstruktor kopii działa w sposób płytki (kopiowanie przez wartość)

Dla przykładu:

```
MyString b = a;
```

Działa tak:

b.str=a.str; //będzie źle, potem zwolnimy destruktorem dwa razy ten sam obszar pamięci !

b.len = a.len;//będzie dobrze

By rozwiązać ten problem należy dokonać głębokiej kopii. Nie kopiujemy tylko adresu ciągu, a cały ciąg znaków.

Poprawna implementacja konstruktora kopii wygląda w sposób następujący:

```
MyString::MyString(const MyString& s)
{
	cout << "Jestem konstruktorem kopiującym" << endl;
	len = s.len;
	str = new char[len + 1];
	strcpy_s(str, len + 1, s.str);
}
```

Przetestuj ponownie:

```
MyString a("aaaaaa");
MyString b = a;
MyString c(a);
MyString *d =new MyString(a);
test1(a);
test2(a);
```

5. Operator przypisania.

Przetestuj poniższy fragment kodu:

```
MyString a("aaaaaa");
MyString b;
a = b;
```

Problem jest analogiczny jak w przypadku konstruktora kopii – płytkie kopiowanie. Należy przeciążyć operator przypisania z uwzględnieniem głębokiej kopii. Implementacja przedstawia się następująco:

```
MyString & MyString::operator=(const MyString& st)
{
	if (this == &st) //kontrola autoprzypisania jak ktoś napisze a=a;
	{
		return *this;
	}
	delete[]str; //zwalniamy poprzedni ciąg
	len = st.len;
	str = new char[len + 1];
	strcpy_s(str, len + 1, st.str);
	return *this;
}
```

Przetestuj ponownie fragment:

```
MyString a("aaaaaa");
MyString b;
a = b;
```

6. Przygotuj implementację operatorów <, > oraz == z wykorzystaniem funkcji strcmp.

7. Przygotuj implementację operatora [] zwracającego wartość pod indeksem w tablicy.

8. Załóżmy, ze chcemy dokonać następującej operacji:

```
MyString a;
char temp[30];
cin.getline(temp, 30);
a = temp;
```

Ile razy wywoła się destruktor ?

Przygotuj odpowiednie przeciążenie operatora=.

Powinno wyglądać w sposób następujący:

```
MyString & MyString::operator=(const char* st)
{
        delete[]str; //zwalniamy poprzedni ciąg
        len = strlen(st);
        str = new char[len + 1];
        strcpy_s(str, len + 1, st);
        return *this;
}
```

9. Przygotuj dynamiczną tablicę obiektów klasy MyString. Wyświetl je w pętli z wykorzystaniem operatora <<.
   Pamiętaj o zwolnieniu pamięci dla tej tablicy. Zwróć na liczbę wywołań destruktora.

10. Przeciąż operator+ który umożliwi łączenie dwóch ciągów w jeden.

11. Przygotuj funkcję konwersji rzutującą obiekt klasy MyString na liczbę całkowitą równą długości ciągu znaków.