Wrocław 2024.12.30

Autorzy: Michał Przewoźniczek i Marcin Komarnicki

Techniki Efektywnego Programowania – zadanie poprawkowe
Literały definiowane przez użytkownika (C++11 i wyżej)

Literały używane są w kodzie do podawania wartości wprost. Na przykład, 123 jest literałem całkowitoliczbowym reprezentującym liczbę całkowitą 123, ‘a’ jest literałem znakowym reprezentującym znak a, 21.5 jest literałem zmiennopozycyjnym reprezentującym liczbę rzeczywistą 21,5, „ala” jest literałem napisowym reprezentującym napis ala oraz true jest literałem typu logicznego reprezentującym prawdę logiczną.

Każdy literał ma swój określony typ. Na przykład, typem literału całkowitoliczbowego 123 jest `int`, natomiast typem literału 21.5 jest `double`. Literały używane są m.in. do przypisywania wartości zmiennych (`int i = 0;`), gdzie do zmiennej `i` typu `int` przypisujemy literał `0`, który przechowuje wartość 0. Jeżeli chcemy użyć literału innego typu niż domyślny to w wielu przypadkach musimy dodać odpowiednią literę lub litery na koniec literału. Na przykład, literał całkowitoliczbowy 123L i literał zmiennopozycyjny 21.5F są typu `long int` i `float` zamiast `int` i `double`. Użycie dodatkowych znaków w literale może także określać użyty system liczbowy lub jednostkę miary. Na przykład, literał całkowitoliczbowy 0123 oznacza liczbę całkowitą zapisaną w systemie ósemkowych, więc ten literał przechowuje wartość 83.

Od standardu C++11 użytkownicy mogą definiować własne literały. W celu uniknięcia konfliktów możemy jedynie rozszerzać istniejące literały używając przyrostków rozpoczynających się od znaku podkreślenia. W ramach niniejszego zadania, wystarczająca jest umiejętność definiowania własnych literałów rozszerzających literały całkowitoliczbowe i zmiennopozycyjne. Deklaracja własnego literału rozszerzającego literał całkowitoliczbowy ma postać:

```
<typ> operator""_<nazwa>(unsigned long long <argument>);
```

podobnie jak deklaracja własnego literału rozszerzającego literał zmiennopozycyjny:

```
<typ> operator""_<nazwa>(long double <argument>);
```

Przykład dla klasy przechowującej temperaturę w stopniach Celsjusza.

```
class CTemperature
{
public:
	CTemperature(double dValueInCelsius);

	CTemperature operator+(const CTemperature& cOther);

private:
	double d_value_in_celsius;
};
```

```
CTemperature::CTemperature(double dValueInCelsius)
	: d_value_in_celsius(dValueInCelsius)
{
}

CTemperature CTemperature::operator+(const CTemperature& cOther)
{
	return CTemperature(d_value_in_celsius + cOther.d_value_in_celsius);
}
```

Dla tak zaimplementowanej klasy możemy zdefiniować trzy własne literały, które w czytelny sposób pozwolą nam m.in. dodawać temperatury w różnych jednostkach.

```
CTemperature operator""_celsius(long double dTemperatureInCelsius)
{
	return CTemperature(dTemperatureInCelsius);
}

CTemperature operator""_fahrenheit(long double dTemperatureInFahrenheit)
{
	double d_temperature_in_celsius = (dTemperatureInFahrenheit - 32) * 5 / 9;
	return CTemperature(d_temperature_in_celsius);
}

CTemperature operator""_kelvin(unsigned long long iTemperatureInKelvin)
{
	double d_temperature_in_celsius = iTemperatureInKelvin - 273;
	return CTemperature(d_temperature_in_celsius);
}
```

Powyższe literały pozwalają nam na zapisanie następującego fragmetnu kodu.

```
CTemperature c_temperature = 87.8_fahrenheit + 300_kelvin + 12.5_celsius;
```

Zadanie
UWAGI:

1. **Pisząc własny program można użyć innego nazewnictwa niż to przedstawione w treści zadania i w przykładach. Należy jednak użyć jakiejś spójnej konwencji kodowania, zgodnie z wymaganiami kursu.**
2. **Nie wolno używać wyjątków (jest to jedynie przypomnienie, wynika to wprost z zasad kursu).**
3. **Wolno używać wyłącznie komend ze standardu C++98 (wyjątkiem są literały definiowane przez użytkownika).**

1. Oprogramuj klasę CTimeSpan reprezentującą przedział czasowy (okres). Powinna ona uwzględniać dni, godziny i minuty.

2. Oprogramuj dodawanie i odejmowanie obiektów klasy CTimeSpan.

3. Zaimplementuj własne literały dla dni, godzin i minut. Na przykład, 12_d może oznaczać 12 dni, 3_h może oznaczać 3 godziny, natomiast 30_m może oznaczać 30 minut.

4. Oprogramuj klasę CDateTime reprezentującą dzień, miesiąc, rok, godzinę i minutę.
   Uwagi:
   4.1. Należy uwzględnić istnienie lat przestępnych.
   4.2. Punkt w czasie reprezentowany przez obiekt klasy CDateTime należy przechowywać jako jedną liczbę zmiennoprzecinkową.
   4.3. Klasa CDateTime nie może stanowić opakowania dla istniejących już klas.

5. Oprogramuj porównywanie obiektów klasy CDateTime (<, <=, >, >=, ==, !=).

6. Zaimplementuj dodawanie/odejmowanie obiektu klasy CTimeSpan do/od obiektu klasy CDateTime.

7. Zaimplementuj odejmowanie dwóch obiektów klasy CDateTime, którego wynikiem jest obiekt klasy CTimeSpan.

Zalecana literatura
Materiały możliwe do znalezienia w Internecie