Zad. 4: Szablonu dla układu równań liniowych

1 Cel ćwiczenia

Wykształcenie zdolności definiowania szablonów funkcji i klas oraz abstrahowania operacji arytmetycznych od konkretnych typów. Unaocznienie problemów związanych z programowaniem uogólnionym i potencjalnych zalet takiego podejścia.

2 Opis zadania programowego

Należy przerobić program rozwiązujący układ równań z poprzedniego zadania, tak aby był w stanie rozwiązać układ równań liniowych z pięcioma niewiadomymi zarówno dla przypadku liczb rzeczywistych, jak też zespolonych. W tym celu należy przerobić moduły klas `Wektor`, `Macierz`, `UkladRownanLiniowych` na szablony. Mają one być parametryzowane zarówno wartością rozmiaru (ilością niewiadomych), jak też typem. Wspomniany typ ma być typem współrzędnych wektorów oraz współczynników macierzy, jak też typem szukanych wartości niewiadomych.

Niniejsze zadanie jest swego rodzaju sprawdzianem poprawności przyjętych wcześniej koncepcji. Im były one lepsze tym mniej problemów będzie nastręczała zamiana rozmiaru i typów liczb, na których działa program.

2.1 Działanie programu

Program ma umożliwiać dwa warianty obliczeń, tzn. rozwiązywanie układu pięciu równań liniowych z pięcioma niewiadomymi dla przypadku, gdy współczynniki są liczbami rzeczywistymi, oraz dla przypadku liczb zespolonych. Wybór wariantu obliczeń ma być realizowany na początku poprzez podanie znaku `'r'` (obliczenia dla wariantu liczb rzeczywistych) lub `'z'` (obliczenia dla wariantu liczb zespolonych). Podanie jakiegokolwiek innego znaku powinno skutkować przerwaniem działania programu ze stosownym komunikatem.

Program nie ma udostępniać żadnego dodatkowego interfejsu użytkownika. Zakłada się, że po wyborze wariantu działanie programu ma przebiegać analogicznie jak w przypadku wcześniejszego zadania. Jedyną różnicą jest to, że tym razem program nie musi wyświetlać normy wektora błędu (jego długości).

Dla ułatwienia testowania programu, należy skorzystać z plików testowych, które zawierają dane dla wariantu liczb rzeczywistych i zespolonych. Ich zawartością jest to, co należy wprowadzić z klawiatury. Dzięki temu przy wywołaniu programu wystarczy przekierować jego wejście standardowe.

2.2 Wymagania co do konstrukcji programu

Oprócz wymagań sformułowanych w opisie zadania należy uwzględnić uwarunkowania przedstawione poniżej.

- Program musi zachować strukturę modułową i odpowiednią strukturę kartotek. O ile będzie to konieczne, należy zmodyfikować plik `Makefile` (np. gdy dodany zostanie nowy moduł/plik nagłówkowy).

- Wszystkie metody, które nie zmieniają stanu obiektu, na którym działają, powinny być metodami typu **const**.
- Wszystkie klasy i metody powinny być opisane.

Oprócz tego pozostają w mocy wszystkie wcześniejsze wymagania dotyczące struktury katalogów, pliku Makefile, modułowej struktury programu, jak też opisów.

3 Przykład działania programu

Dla ułatwienia sobie pracy warto zapisać dane w pliku i później czytać dane z pliku poprzez przekierowanie go na wejście standardowe programu. Taki sposób postępowania przedstawiony jest poniżej.

```
jkowalsk@noxon: rozwiazanie> cat rownanie_liczb_rzeczywistych.dat
    r
    2.00     1.00     1.00     1.00     2.00
    2.00     2.00     3.00     1.00     2.00
    1.00     1.50     1.00     1.00     0.00
    3.00     1.00     2.00     4.00     0.00
    3.00     2.00     2.00     0.00     1.00
    9.00     8.00     8.00     9.00     1.00

jkowalsk@noxon: rozwiazanie> ./uklad_rownan < rownanie_liczb_rzeczywistych.dat

Uklad rownan liniowych o wspolczynnikach rzeczywistych

Macierz A^T:
    2.00     1.00     1.00     1.00     2.00
    2.00     2.00     3.00     1.00     2.00
    1.00     1.50     1.00     1.00     0.00
    3.00     1.00     2.00     4.00     0.00
    3.00     2.00     2.00     0.00     1.00
    9.00     8.00     8.00     9.00     1.00

Wektor wyrazow wolnych b:
    9.00     8.00     8.00     9.00     1.00

Rozwiazanie x = (x1, x2, x3, x4, x5):

   -0.13     0.47     3.51     1.29     0.31

 Wektor bledu: Ax-b  = (-1.78e-15  -1.78e-15  -1.78e-15  0.00e+00 -2.89e-15)

jkowalsk@noxon: rozwiazanie>_
jkowalsk@noxon: rozwiazanie> cat rownanie_liczb_zespolone.dat
    z
    (2.00+1.00i) (1.00+1.00i) (1.00+1.00i) (1.00+2.00i) (2.00+3.00i)
    (2.00+3.00i) (2.00+5.00i) (3.00+7.00i) (1.00+2.00i) (2.00+2.00i)
    (1.00+2.00i) (1.50+3.00i) (1.00+1.00i) (1.00+1.00i) (0.00+2.00i)
```

```
    (3.00+1.00i) (1.00+4.00i) (2.00+1.00i) (4.00+3.00i) (0.00+0.00i)
    (3.00+2.00i) (2.00+2.00i) (2.00+0.00i) (0.00+1.00i) (1.00+1.00i)
    (9.00+1.00i) (8.00+2.00i) (8.00+8.00i) (9.00+5.00i) (1.00+1.00i)

jkowalsk@noxon: rozwiazanie> ./uklad_rownan < rownanie_liczb_zespolone.dat

Uklad rownan liniowych o wspolczynnikach zespolonych

Macierz A^T:
    (2.00+1.00i) (1.00+1.00i) (1.00+1.00i) (1.00+2.00i) (2.00+3.00i)
    (2.00+3.00i) (2.00+5.00i) (3.00+7.00i) (1.00+2.00i) (2.00+2.00i)
    (1.00+2.00i) (1.50+3.00i) (1.00+1.00i) (1.00+1.00i) (0.00+2.00i)
    (3.00+1.00i) (1.00+4.00i) (2.00+1.00i) (4.00+3.00i) (0.00+0.00i)
    (3.00+2.00i) (2.00+2.00i) (2.00+0.00i) (0.00+1.00i) (1.00+1.00i)

Wektor wyrazow wolnych b:
    (9.00+1.00i) (8.00+2.00i) (8.00+8.00i) (9.00+5.00i) (1.00+1.00i)

Rozwiazanie x = (x1, x2, x3, x4, x5):

   (-0.13-4.66i) (0.47+5.50i) (3.51-1.45i) (1.29-3.62i) (0.31+2.95i)

 Wektor bledu: Ax-b  = ((-1.78e-15-1.78e-15i) (-1.78e-15+3.55e-15i) (-1.78e-15-8.88e-16i) (0.00e+

jkowalsk@noxon: rozwiazanie>_
```

Ułożenie znaków w linii w przypadku wyświetlania macierzy, nie musi być idealnie równe, aby jeden wiersz długością pasował do drugiego. Format wyświetlania liczb również nie musi być w pełni zgodny z tym, który jest w przykładzie.

4 Zalecenia (ważne)

Istniejące konstrukcje w języku C++ pozwalają na niejawne konwersje typów. Jeżeli mechanizmy nie są świadomie stosowane lub nie są dobrze zrozumiane, to mogą prowadzić do błędów działania programu, które będą bardzo trudne do zdiagnozowania. Aby tego uniknąć należy zastosować zalecenia przedstawione poniżej.

<u>Nie</u> zaleca się definiowanie konstruktora jednoargumentowego postaci:

```
LZespolona::LZespolona(float r);
```

Zalecane jest natomiast zdefiniowanie konstruktora bezparametrycznego oraz dwuargumentowego, tzn.

```
LZespolona::LZespolona();
LZespolona::LZespolona(float r, float i);
```

5 Przeróbka klasy `Wektor` na szablon – ćwiczenie

W niniejszym zadaniu należy przerobić trzy klasy na szablony. Nieumiejętne podejście do tego zadania prowadzi do wielu komplikacji i problemów na poziomie kompilacji. Ich złe rozwiązanie może spowodować powstawania błędów, które będą trudne do zlokalizowania. Z tych też względów należy do tego podejść stopniowo i systematycznie. Przeróbki należy rozpocząć od podstawowej klasy (tzn. `Wektor`). Należy je prowadzić w osobnej kartotece. Przykład znajduje się w kartotece

```
~bk/edu/kpo/zad/z4/od-klasy-do-szablonu
```

W kartotece tej znajdują się 3 podkartoteki, które obrazują proponowany sposób przeróbki składający się z trzech etapów (kroków), które przedstawione są w dalszej części.

Krok 1: Wydzielamy moduł, który ma zostać przerobiony na szablon i tworzymy plik `main.cpp`, w którym znajdują podstawowe testy metod i funkcji udostępnianych przez moduł. Przykładowa postać plików, które rezultatem tego etapu, przedstawiona jest w kartotece `krok1-jeszcze-klasa`.

Krok 2: Tworzymy plik nagłówkowy, w którym zdefiniujemy szablon klasy. Chcąc uniknąć przypadkowych błędów, lepiej jest nadać szablonowi nieco inną nazwę niż oryginalna klasa, np. `SWektor`.

W pliku nagłówkowym umieszczamy w nim stopniowo poszczególne elementy *przerabianej* klasy. W trakcie pracy nie usuwamy starego modułu. W pliku `main.cpp` dopisujemy dołączenie pliku z szablonem poprzez dyrektywę `#include`, np.

```
#include "SWektor.hh"
```

Na początkowym etapie wystarczy, że definicja szablony klasy `SWektor` zawierać będzie tylko pola bez metod, np.

```
template <typename STyp, int SWymiar>
class SWektor {
    STyp  _Wsp[SWymiar];
  public:
};
```

Natomiast w funkcji `main` wystarczy umieścić tylko definicję zmiennej nowego typu, np.

```
SWektor<double,ROZMIAR>   WekTestowy;
```

Po doprowadzeniu do poprawnej kompilacji możemy przejść do dodawania kolejnych elementów definicji szablonu. Wreszcie po przeniesieniu definicji wszystkich niezbędnych metod możemy usunąć moduł klasy `Wektor` oraz odpowiednio *wyczyścić* funkcję `main`, jak też zmodyfikować plik `Makefile`. Finalna postać tego etapu przedstawiona jest w kartotece `krok2-juz-szablon`.

Krok 3: W końcowym kroku pozostaje sprawdzić, czy będzie możliwe użycie tak zdefiniowanego szablonu w przypadku, gdy typem pól będzie klasa modelująca pojęcie liczby zespolonej. W tym celu należy wykorzystać moduł, który został stworzony w ramach zadania nr 2. Prawie na pewno w strukturze `LZespolona` będzie potrzeba dodatkowego zdefiniowania przeciążenia operacji podstawienia liczby rzeczywistej (tzn. wartości typu `double`). Przykład realizacji tego przeciążenia znajduje się w katalogu `krok3-wektor-zespolony`. W tej też kartotece znajduje się końcowy rezultat realizacji tego kroku.

Po zakończeniu pracy nad szablonem `SWektor`, należy dodać definicję szablonu `SMacierz` i analogicznie do kroku 2. dla szablonu `SWektor` stopniowo przenosić defincje pól i metod, tak aby było poprawne testowe utworzenie obiektu klasy `SMacierz<double,ROZMIAR>` oraz wywołanie dopisywanych definicji poszczególnych metod. Następnie, analogicznie do kroku 3, dodajemy utworzenie instancji `SMacierz<LZespolona,ROZMIAR>` i modyfikujemy/rozszerzamy definicję struktury `LZesplona`, aby osiągnąć poprawność kompilacji i konsolidacji oraz realizowanych testów. Analogicznie postępujmy z klasą `UkladRownanLiniowych`.

W ramach ćwiczenia realizowanego na zajęciach należy przekopiować zawartość kartoteki `od-klasy-do-szablonu` i wzorując się na przykładzie należy rozszerzyć szablon `SWektor` znajdujący się w podkartotece `krok2-juz-szablon` o wszystkie metody, które są w klasie `Wektor` stworzonej w ramach poprzedniego zadania.

6 Wersje zadania

W dalszej części przedstawione zostały propozycje wersji zadania zarówno łatwiejsze, jak też ciekawszej :)

6.1 Wersja łatwiejsza

W poprzednim zadaniu w ramach łatwiejszej wersji można było zrealizować zadanie dla układu z dwoma zmiennymi. Program ten w takiej postaci można przekształcić do wersji z szablonami jako wersję łatwiejszą.

6.2 Wersja ciekawsza – rozszerzenie nieobowiązkowe

Podobnie jak w poprzedniej wersji zadania, należy wyświetlić długość wektora błędu. Do wyliczania długości powinna zostać zdefiniowana metoda w szablonie `SWektor`. Dla współrzędnych typu `double` długość liczona jest jako pierwiastek z sumy kwadratów wartości poszczególnych współrzędnych. W przypadku liczb zespolonych trzeba uwzględnić, że nie mogą to być kwadraty, a iloczyny liczb przez ich sprzężenia. Problem ten można i należy rozwiązać definiując specjalizację metody `Dlugosc` szablonu `SWektor` dla typu `LZespolone`. Więcej o specjalizacji będzie powiedziane na wykładzie.

7 Przygotowanie do zajęć

7.1 Tydzień 0

Ze względu na to, że zadanie niniejsze bazuje na wynikach zadania wcześniejszego, nie jest wymagane dodatkowe przygotowanie do zajęć. Jednak w ramach zajęć należy zdefiniować

zasadniczą cześć szablonu `SWektor`, który będzie podstawą do tworzenia dalszych szablonów. Zalecany sposób definiowania szablonu, który bazuje na wcześniejszej definicji klasy `Wektor`, został opisany w rozdziale 6. Pragnąc ułatwić sobie pracę należy skorzystać z dostarczonych przykładów zawartych w kartotece ∼`bk/edu/kpo/zad/z4`.

7.2 Tydzień 1

Przed zajęciami musi zostać zdefiniowany szablon `SMacierz` oraz elementarne testy w funkcji `main`, które będą pokazywały efekt pracy programu, gdy tworzona jest instancja klasy macierzy dla typu `double` oraz dla `LZesplona` (patrz przykład dla szablonu `SWektor` w podkartotece `krok3-wektor-zespolony`). Szablon klasy `SMacierz` musi być zdefiniowany w osobnym pliku nagłówkowym. Należy również odpowiednio zmodyfikować/uzupełnić opisy, o ile okaże się to konieczne. Wszystko co będzie ponad to będzie oceniane *in plus*.

7.3 Tydzień 2

Rozliczenie się z gotowego programu i rozpoczęcie następnego zadania (tydzień 0 dla zadania nr 5).