# Zad. 5: Sterowanie robotem mobilnym

## 1 Cel ćwiczenia

Wykształcenie umiejętności modelowania kluczowych dla danego problemu pojęć. Tworzenie diagramu klas, czynności oraz przypadków użycia. Wykorzystanie dziedziczenia do modelowanie kilku różnych pojęć będących uszczegółowieniem bardziej ogólnego pojęcia. Wykorzystanie pól i metod statycznych.

## 2 Program zajęć

- *Ocena realizacji zadania z poprzedniego laboratorium* – ocenie podlega poprawność realizacji zadania, styl pisania programu oraz opisy.
- *Modyfikacja programu wg wskazań osoby prowadzącej* – ocenie będzie podlegała poprawność realizacji modyfikacji. Pracę nad modyfikacją programu (wszystkie operacje należy wykonywać na kopii) należy rozpocząć już w trakcie pierwszej fazy laboratorium, gdyż prowadzący nie będzie w stanie ocenić wcześniejszego programu wszystkim jednocześnie.
- *Realizacja wstępnej fazy prac nad nowym zadaniem* – w ramach wstępnej realizacji zadania należy wypisać pojęcia, które będą musiały zostać zamodelowane poprzez odpowiednie klasy. Wskazane jest stworzenie diagramu klas w języku UML. Należy również zdefiniować wybrane klasy modelujące podstawowe pojęcia i doprowadzić do poprawnej kompilacji i konsolidacji programu.
  ***Uwaga: Menu programu tworzymy na samym końcu!***
- *Ocena realizacji wstępnej fazy zadania*

## 3 Opis zadania programowego

Należy napisać program, który zwizualizuje położenie robota mobilnego na scenie roboczej (w tym celu należy wykorzystać moduł `lacze_do_gnuplota`). Program powinien umożliwiać zadawanie przez użytkownika obrotu robota i jego wizualizację, jak też ruchu na wprost na zadaną odległość. W celu interakcji z użytkownikiem program powinien udostępniać proste menu, za pomocą którego użytkownik powinien móc wykonywać następujące czynności:

- zmianę orientacji robota,
- zadanie ruch na wprost na zadaną odległość (wykonanie tej operacji powinno być wizualizowane rysunkiem ścieżki pokonanej przez robota)
- wyświetlenie menu,
- zakończenie działania programu.

Dodatkowo program powinien wyświetlać wraz z menu informację o liczbie aktualnie istniejących obiektów klasy `Wektor2D` oraz łączną liczbę wszystkich obiektów klasy `Wektor2D` utworzonej do tej pory.

# 4 Przygotowanie do zajęć

Należy przygotować diagram klas proponowanych struktur danych.

# 5 Wymagania co do konstrukcji programu

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

- Należy zauważyć, że wszystkie obiekty w programie, takie jak robot i ścieżka (a w późniejszym zadaniu również przeszkoda), mają mieć swoją reprezentację graficzną. Obrys robota jest łamaną zamkniętą, zaś rysunek ścieżki łamaną otwartą. Należy zaprojektować odpowiednią hierarchię dziedziczenia, która pozwoli wydzielić model bardziej ogólnego pojęcia. Klasa, która będzie modelowała to pojęcie, będzie wspólną częścią klas modelujących pojęcie robota i ścieżki. Będą one tym samym specjalizacją tego bardziej ogólnego pojęcia.

- Zbiór wierzchołków łamanych może mieć różną wielkość dla różnych przypadków. Dlatego też, aby móc to uwzględnić, należy wykorzystać szablon `vector<>`.

- Należy również zwrócić uwagę, że w przypadku rysunku robota, aby nie dopuścić do akumulacji błędów obliczeń związanych z obrotami, należy przechowywać współrzędne obiektu *wzorcowego* sprzed transformacji.

- Należy zwrócić uwagę na to, że w opisie programu występuje pojęcie *sceny*. Powinno ono mieć swoje odzwierciedlenie w strukturze danych programu.

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

- Każda z klas powinna zostać zdefiniowana w oddzielnym pliku nagłówkowym. Metody tej klasy powinny być natomiast definiowane w osobnym module związanym z daną klasą, np. definicja klasy `Sciezka` powinna znaleźć się w pliku nagłówkowym `Sciezka.hh`, zaś metody w pliku `Sciezka.cpp`. Proste metody można definiować bezpośrednio w ciele klasy.

- Dla poszczególnych klas należy przeciążyć niezbędne operatory działające na strumieniach. Nie wszystkie przeciążenia są w tym zadaniu potrzebne. Na pewno będą potrzebne przeciążenia operatorów wczytywania i zapisu dla klasy `Wektor2D`.

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

- Program powinien umożliwiać graficzną wizualizację wyników działania. Pozwala na to moduł łącza do programu `gnuplot`.

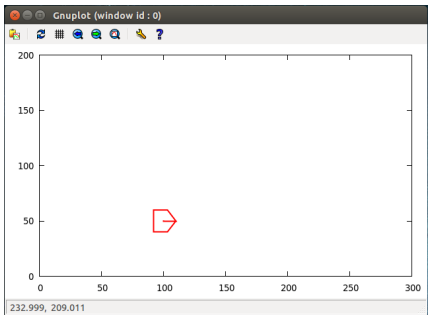
- Zarówno obrót jak też ruch robota powinien być animowany, tzn. obrót jak też ruch powinien być rozbity na sekwencję mniejszych obrotów lub ruchów w przód.

- Wszystkie klasy i metody oraz funkcje powinny zostać opisane. Opis ten powinien być zgodnie z wymogami systemu `doxygen`. Ponadto należy wygenerować dokumentację w formacie HTML za pomocą programu `doxygen`.

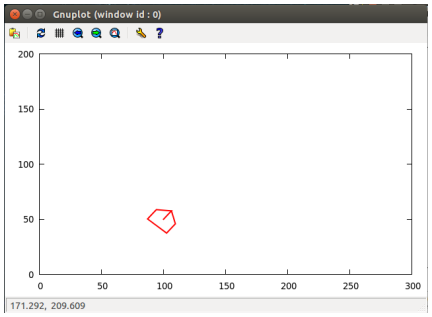
Oprócz tego pozostają w mocy wszystkie wcześniejsze wymagania dotyczące struktury katalogów, pliku Makefile, modułowej struktury programu, jak też opisów.

# 6 Przykład działania programu

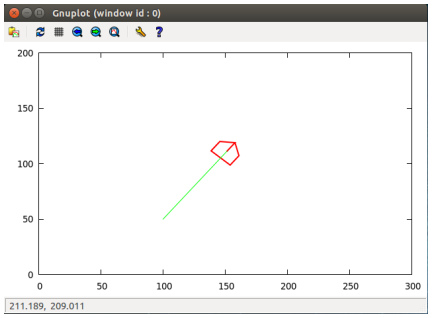
```
jkowalsk@panamint> ./robot_na_scenie

 Laczna ilosc stworzonych obiektow klasy Wektor2D: 184
       Ilosc istniejacych obiektow klasy Wektor2D: 26

  o - obrot robota
  j - jazda na wprost
  w - wyswietl ponownie menu

  k - zakoncz dzialanie programu

 Twoj wybor (w - wyswietl menu)> o
```



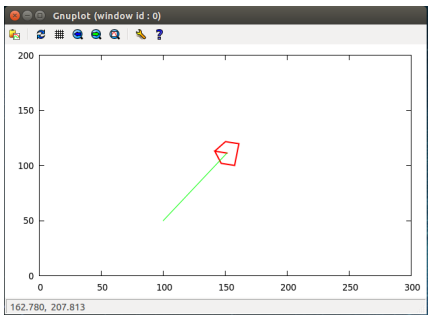

```
 Podaj wartosc kata obrotu robota w stopniach.
 Kat obrotu: 50

 Twoj wybor (w - wyswietl menu)>j
```



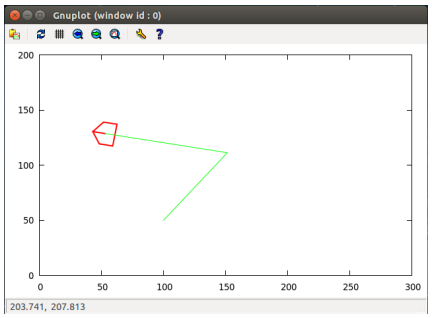

```
 Podaj dlugosc drogi ruchu robota na wprost.
 Dlugosc drogi: 80

 Twoj wybor (w - wyswietl menu)> o
```



```
Podaj wartosc kata obrotu robota w stopniach.
Kat obrotu: 120

Twoj wybor (w - wyswietl menu)> j
```




```
Podaj dlugosc drogi ruchu robota na wprost.
Dlugosc drogi: 100

Twoj wybor (w - wyswietl menu)> t
```




```
Twoj wybor (w - wyswietl menu)> k

 Koniec dzialania programu.

jkowalsk@panamint> _
```

# 7 Materiały pomocnicze

W ramach materiałów pomocniczych został dostarczany żaden zalążek, który zawiera moduł `lacze_do_gnuplota` i pokazuje jego przykładowe wykorzystanie. Zawiera też wstępną definicję klasy `ObiektGraficzny` oraz wykorzystuje instancję szablonu `SWektor` z parametrami odpowiednio `double` i 2. Do instancji tej stworzony jest alias `Wektor2D` poprzez polecenie `typedef`. Zalążek ten zawiera również utworzony plik konfiguracyjny dla programu `doxygen` (plik ten znajduje się w podkatalogu `dox`), który pozwala na wygenerowanie dokumentacji programu. Proponuje się zacząć pracę nad programem wykorzystując dostarczony zalążek.

Na `panamincie` w podkartotece `~bk/edu/kpo/zad/z5` można znaleźć skrypt, który uruchamia program demonstrujący przykładową realizację zadania. Skrypt ten można uruchomić poleceniem:

```
~bk/edu/kpo/zad/z5/przyklad/przyklad_rozwiazania.sh
```

Ze względu na to, że program tworzy okienko graficzne, w przypadku połączenia z panamintem za pomocą programu `ssh` należy pamiętać, aby użyć opcji `-X`.

# 8 Rozszerzenia

Możliwych jest kilka wariantów rozszerzenia tego zadania.

## 8.1 Mniej ambitne rozszerzenia

### 8.1.1 Skalowanie obrysu robota

Kształt robota powinno być można skalować. Tak więc jego obrys użytkownik powinien mieć możliwość powiększania lub pomniejszania.

### 8.1.2 Zmiana pozycji i szybkości ruchu robota

Użytkownik powinien móc zadać pozycję robota. Powinin również mieć możliwość zmiany szybkości ruchu na wprost (długości elementarnego kroku w trakcie ruchu).

## 8.2 Bardziej ambitne rozszerzenie

### 8.2.1 Rozszerzenie grafiki

Zamiast rysowania 2D należy rysować robota oraz jego ścieżkę ruchu wykorzystując grafikę 3D.

# 9 Uproszenia

- Można zrezygnować z animacji ruch i obrotu robota. Wykorzystanie tego uproszczenia powoduje obniżenie oceny o $0,5$.

# 10 Wymagania i zarys programu zajęć w okresie realizacji zadania

Przystępując do pracy nad programem należy pamiętać, że menu programu dodajemy na samym końcu, gdy stworzymy już i przetestujemy wszystkie niezbędne funkcjonalności. Zaczynanie pracy od menu nie jest dobrym rozwiązaniem.

## 10.1 Tydzień 0

Nie jest wymagane dodatkowe przygotowanie. Niemniej wskazane jest wstępne zaprojektowanie struktur danych i stworzenie odpowiedniego diagramu klas.

W trakcie zająć należy zapoznać się z dostarczonym załążkiem i przenieść do niego własny szablon klasy `SWektor`. Jeśli jest on stworzony pod inną nazwą, to po jego przeniesieniu należy zmodyfikować plik `Wektor2D.hh`, w którym jest odwołanie do tego szablonu. Należy również zmodyfikować plik `Makefile`, aby zastąpić nazwę pliku `SWektor.hh` nazwą własnego szablonu.

## 10.2 Tydzień 1

Przed zajęciami muszą zostać przygotowane następujące elementy zadania. Wszystko co będzie ponad to będzie oceniane *in plus* (oprócz menu programu). W tej wersji programu nie jest pożądane, aby występowało menu. Wyjątkiem jest sytuacja, gdy program zostanie wcześniej skończony.

- Pełny diagram klas w wersji elektronicznej.
- Powinny być wstępnie zdefiniowane klasy `Scena`, `Robot` oraz `Sciezka`. Należy zwrócić uwagę, że pojęcia `Robot` i `Sciezka` w tym zadaniu wiążą się z odpowiednią reprezentacją graficzną. To jest element wspólny tych pojęć. Tak więc należy wydzielić bardziej ogólne pojęcie, które zostanie zamodelowane jako klasa bazowa w stosunku do klas modelujących pojęcia takie jak: `Robot` oraz `Sciezka`. Wszystkie klasy muszą mieć zdefiniowane podstawowe metody.
- Warunkiem koniecznym pozytywnej oceny jest poprawna kompilacja. W trakcie kompilacji nie powinny być generowane żadne ostrzeżenia.
- W funkcji `main` powinien być kod, który demonstruje rysowanie robota. Wspomniany kod powinien demonstrować również ruch robota na wprost. Ścieżka ruchu robota nie musi być rysowana.
  **Uwaga:** W tej wersji programu nie powinno być jeszcze menu.
- Wszystkie klasy i metody muszą być opisane oraz powinna być generowana dokumentacja za pomocą programu `doxygen`.

## 10.3 Tydzień 2

Rozliczenie się z gotowego programu i rozpoczęcie następnego zadania.