Middle East Technical University | Department of Computer Engineering

# CENG 242

## Programming Language Concepts

Spring 2020-2021

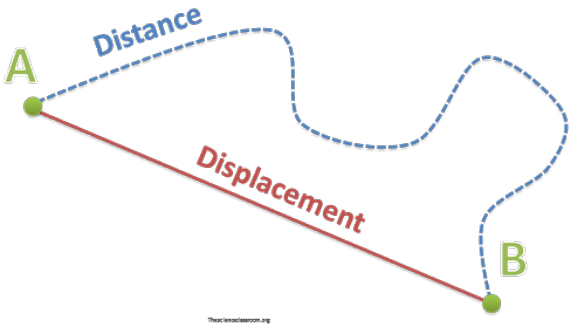
## Programming Exam 5

Due date: 27 May 2021, Thursday, 23:59

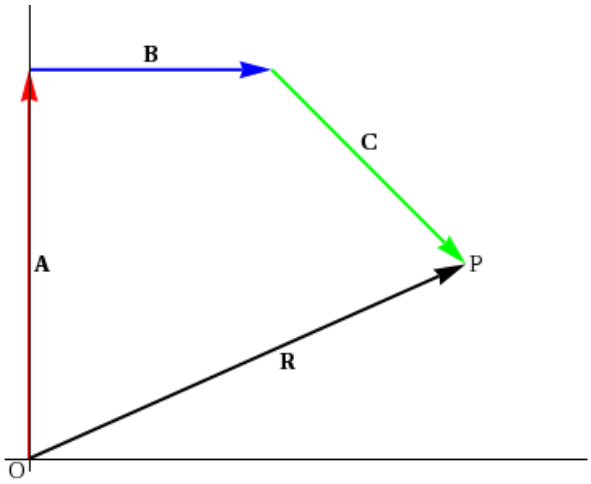
# 1 Problem Definition

In your first programming examination for C++, you will be dealing with keeping track of paths consisting of 2D vectors as coordinates. The path will start from (0,0) coordinate and you will be given a series of vectors to add to the end of the path and calculate the new distance and displacement of the path.

Displacement is defined to be the change in position of an object. It is the distance between the final position and the first position of the object. On the other hand, the distance of a path is the total length of the path.




(a) Distance vs. Displacement




(b)
Distance $= |A| + |B| + |C|$
Displacement $= |R|$

Figure 1: Difference between distance and displacement of a path.

## 1.1 General Specifications

- The signatures of the functions, their explanations and specifications are given in the following section. Read them carefully.
- Make sure that your implementations comply with the function signatures.
- You may define any number of helper function(s) as you need.
- You are not allowed to import any *extra* library for this exam.
- You should NOT make any change in header file. The one given to you will not be used in evaluation process.

## 1.2 Quick VPL Tips (In case you've forgotten them!)

- Evaluation is fast. If evaluation seems to hang for more than a few seconds, your code is entering an infinite loop or has an abnormal algorithmic complexity. Or you've lost your connection, which is *much* less likely!
- Although the run console does not support keyboard shortcuts, it should still be possible to copy and paste using the right-click menu (Tested on latest versions of Firefox and Chrome).
- Get familiar with the environment. Press the plus shapes button on the top-left corner to see all the options. You can download/upload files, change your font and theme, switch to fullscreen etc. Useful stuff!

# 2 Path Tracker

Path Tracker is a class keeping track of the final coordinate of the path and the distance and the displacement values.

PathTracker Class Properties

```
class PathTracker
{
public:
    int final_x;          // final position on x-axis
    int final_y;          // final position on y-axis
    float displacement;   // final displacement
    float distance;       // final distance
}
```

There are 2 constructors you have to implement. If there is no argument, then PathTracker will start from the origin. If you are given an array of integers, then each couple in the array will be used as a vector and the path will be constructed using the list of couples. Besides, the distance and the displacement values will be updated.

For example, [1, 2, 3, 4, 5, 6] keeps three vectors: [1, 2], [3, 4], [5, 6]. These three vectors will be added end-to-end to construct the path.

PathTracker Class Constructors

```
class PathTracker
{
....

PathTracker();  // start the path from the origin

// number_of_vectors: number of vectors to add end-to-end
// existing_path: number_of_vectors*2 elements representing vectors
// [x1, y1, x2, y2, ....., xn, yn], n=number_of_vectors
PathTracker(int *existing_path, int number_of_vectors);
}
```

There are 5 operator overloading functions to be implemented.

- PathTracker operator+ =(const int *new_vector): takes an integer array with the length of 2, containing x and y components of the vector to be added to the end. During this operation, the final x and y coordinates, displacement and distance values of the path will be updated.
- bool operator==(const PathTracker rhs) const: takes a PathTracker object to compare displacements. If two objects' **displacements** are equal, it will return true else will return false.
- bool operator>(const PathTracker rhs) const: takes a PathTracker object to compare displacements. If the original object's **displacement** is longer, it will return true else will return false.
- bool operator>(const PathTracker rhs) const: takes a PathTracker object to compare displacements. If original object's **displacement** is shorter, it will return true else will return false.
- bool operator==(float distance) const: takes a floating-point number and compare it with the object's **distance**. If they are equal, it will return true else will return false.

PathTracker Class Operator Overloading Functions

```
class PathTracker
{
....

// Adds new vector to the end
// new_vector: keeping vector components [x, y]
PathTracker &operator+=(const int *new_vector);

// Comparing two PathTrackers by displacements

// returns True if their displacements are equal
bool operator==(const PathTracker &rhs) const;

// returns True if original object's displacement is bigger then the
// compared one
bool operator>(const PathTracker &rhs) const;

// returns True if original object's displacement is less then the
// compared one
bool operator<(const PathTracker &rhs) const;
```

```
    // Checks whether the distance of the object is equal to the given
    // floating-point number
    bool operator==(float distance) const;
    }
```

There are 4 small helper functions to use while keeping track of the path.

- float calculate_displacement(): Calculates and returns the displacement of the path.
- void calculate_distance(int x, int y): takes two integer values as x and y components representing the newly added vector and updates the distance of the path.
- float calculate_l2(int x1, int y1, int x2, int y2): takes 4 integer values as 2 points on vector space. It calculates and returns Euclidean distance between two points.
- void summary(): prints the summary of the path as:

```
Final Position: [x, y] Displacement: #dsp# Distance: #dst#
Final Position: [25,30] Displacement: 39.051 Distance: 39.13
```

PathTracker Class Helper Functions

```
    class PathTracker
    {
    ....

    // Calculates and returns displacement value
    float calculate_displacement();

    // Calculates and sets distance value
    void calculate_distance(int x, int y);

    // Calculates Euclidean distance between two points
    float calculate_l2(int x1, int y1, int x2, int y2);

    // Prints the summary of the path
    void summary();
    }
```

# 3 Examples

Example From Scratch and Adding A New Vector to the End

```
int main()
{
    PathTracker pt = PathTracker();
    int coord[2] = {3, 5};
    pt.summary();

    pt += coord;
    pt.summary();

    coord[0] = -3;
    coord[1] = 1;
    pt += coord;
    pt.summary();

    coord[0] = -2;
    coord[1] = -6;
    pt += coord;
    pt.summary();

    return 0;
}

// Output
Final Position: [0,0] Displacement: 0 Distance: 0
Final Position: [3,5] Displacement: 5.83 Distance: 5.83
Final Position: [0,6] Displacement: 6 Distance: 8.99
Final Position: [-2,0] Displacement: 2 Distance: 15.3
```

Example of Existing Path

```
int main()
{
    int existing_path[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    // it is going to add all vectors end-to-end in constructor
    // [1, 2]
    // [3, 4]
    // [5, 6]
    // [7, 8]
    // [9, 10]
    PathTracker pt = PathTracker(existing_path, 5);

    pt.summary();

    return 0;
}

// Output
Final Position: [25,30] Displacement: 39.051 Distance: 39.13
```

Examples of Operator Overloading Functions

```
int main()
{
    int existing_path1[2] = {25, 30};
    int existing_path2[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    PathTracker pt1 = PathTracker(existing_path1, 1);
    PathTracker pt2 = PathTracker(existing_path2, 5);

    pt1.summary();
    pt2.summary();

    std::cout << "pt1 == pt2: " << (pt1 == pt2) << "\n";
    std::cout << "pt1 > pt2: " << (pt1 > pt2) << "\n";
    std::cout << "pt1 < pt2: " << (pt1 < pt2) << "\n";

    // notice that distance can be compared with a floating-point
    // not another PathTracker object.
    std::cout << "distance comparison: " << (pt1 == pt2.distance) <<
       "\n";

    return 0;
}

// Output
Final Position: [25,30] Displacement: 39.051 Distance: 39.051
Final Position: [25,30] Displacement: 39.051 Distance: 39.13
pt1 == pt2: 1
pt1 > pt2: 0
pt1 < pt2: 0
distance comparison: 0
```

# 4 Regulations

1. **Implementation and Submission:** The template files "path_tracker.h", "path_tracker.cpp", and "test.cpp" are available in the Virtual Programming Lab (VPL) activity called "PE5" on `OdtuClass`. At this point, you have two options:
   - You can download the template files, complete the implementation and test it with the given sample I/O on your local machine. Then submit the same file through this activity.
   - You can directly use the editor of VPL environment by using the auto-evaluation feature of this activity interactively. Saving the code is equivalent to submit a file.

   The second one is recommended. However, if you're more comfortable with working on your local machine, feel free to do it. Just make sure that your implementation can be compiled and tested in the VPL environment after you submit it.

   "path_tracker.h" and "test.cpp" files are given to you so that you can work on your local machines. If you choose first option, you have to submit these files as well but they will not be included into evaluation process.

   There is no limitation on online trials or submitted files through OdtuClass. The last one you submitted will be graded.

2. **Programming Language:** You must code your program in C++. Your submission will be compiled with g++ on VPL. You are expected to make sure your code compiles successfully with g++ using the flags -ansi -pedantic.

3. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating (any kind of code sharing and codes taken from internet included) will be punished according to the university regulations.

4. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications. No erroneous input will be used. Therefore, you don't have to consider the invalid expressions.

   **Important Note:** The given sample I/O's are only to ease your debugging process and NOT official. Furthermore, it is not guaranteed that they cover all the cases of required functions. As a programmer, it is your responsibility to consider such extreme cases for the functions. Your implementations will be evaluated by the official testcases to determine your ***actual*** grade after the deadline.