UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS132 - Imperative Programming

Practical 7 Specifications

Release Date: 07-04-2025 at 06:00

Due Date: 11-04-2025 at 23:59

Late Deadline: 13-04-2025 at 23:59

Total Marks: 180

Read the entire specification before starting with the practical.

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you begin coding.***
- This assignment should be completed individually.
- **Every submission will be inspected with the help of dedicated plagiarism detection software.**
- Be ready to upload your assignment well before the deadline. There is a late deadline which is 48 hours after the initial deadline which has a penalty of 20% of your achieved mark. **No extensions will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or structure).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.
- Unless otherwise stated, the usage of C++11 or additional libraries outside of those indicated in the assignment, will **not** be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use C++98**
- All functions should be implemented in the corresponding `cpp` file. No inline implementation in the header file apart from the provided functions.
- The usage of ChatGPT and other AI-Related software to generate submitted code is strictly forbidden and will be considered as plagiarism.

2 Overview

In this practical, you will be exposed to different looping and selection structures. In solidarity with COS 344, who wrote on Saturday, you will implement a simplified terminal-based computer graphics engine.

3 Background

3.1 Implicit Line Equation

A line can be described in various forms. One common form is the *slope-intercept form*:

$$y = mx + c$$

where:

- m is the slope of the line,
- c is the y-intercept.

However, another useful way to define a line is by specifying two points on the line:

$$P_1 = (x_1, y_1), \quad P_2 = (x_2, y_2)$$

Using these two points, we can compute the slope m as:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{(provided } x_2 \neq x_1\text{)}$$

Substituting one of the points (say, P_1) into the slope-intercept form gives:

$$y - y_1 = m(x - x_1)$$

Expanding and rearranging:

$$y - mx + (mx_1 - y_1) = 0$$

This gives us the *implicit form* of the line:

$$Ax + By + C = 0$$

where:

$$A = -(y_2 - y_1), \quad B = x_2 - x_1, \quad C = (y_1(x_2 - x_1)) - (x_1(y_2 - y_1))$$

Putting it all together, we can obtain the following equation:

$$f(x, y) \equiv (y_0 - y_1)x + (x_1 - x_0)y + (x_0y_1 - x_1y_0) = 0 \tag{1}$$

3.2 Computer Graphics Screen

A simple terminal-based graphics engine is a lightweight software library or tool that allows the creation of visual representations, like shapes, animations, or basic games, within a terminal or command-line interface. Instead of using pixels, it typically uses characters (like #, *, or ASCII symbols) to simulate graphics. These engines handle screen refreshing, input handling, and coordinate-based drawing using simple logic and terminal control codes (like ANSI escape sequences). They're often used for retro-style visuals, learning graphics fundamentals, or quick prototyping in environments without GUI support.

A "screen" is defined with coordinate $(0, 0)$ being the origin and at the bottom left of the screen. The maximum number of rows that the screen can have is denoted by n_y, and the maximum number of columns that the screen can have is denoted by n_x. Given that $n_x = 10$ and $n_y = 6$, Figure 1 shows how the screen would be laid out, where each cell represents a pixel:

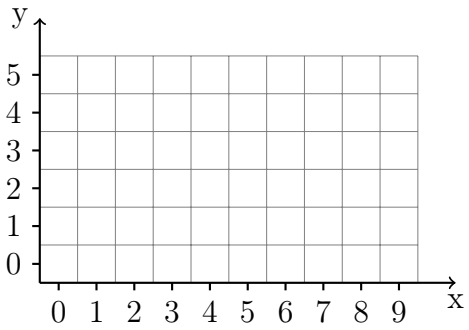

Figure 1: Example of a screen layout with $n_x = 10$ and $n_y = 6$.

3.3 Mid Point Algorithm

A big problem in computer graphics is how to draw a straight line when the line does not perfectly go over the centre of a pixel. Consider Figure 2, which contains two lines we would like to draw.

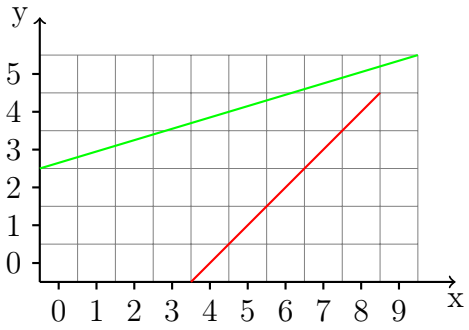

Figure 2: Example of a screen layout with $n_x = 10$ and $n_y = 6$, showing two lines.

When drawing the red line in our terminal screen, we can easily "colour" in the cells that the red line passes through as seen in Figure 3.

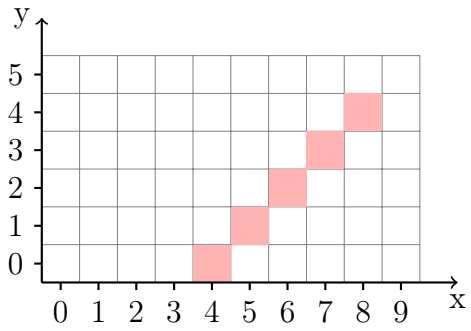

Figure 3: Screen layout with $n_x = 10$ and $n_y = 6$, showing how the pixels will be coloured for the red line.

The green line, on the other hand, is not as easy. This is where the midpoint line drawing algorithm comes into effect. The idea of the midpoint algorithm is that you use the implicit line equation of the line you are trying to draw, to get a "reasonable" resemblance of the line. The precise conditions of the midpoint algorithm will be discussed later. An example of the results obtained from the mid-point algorithm for the green line is given in Figure 4.

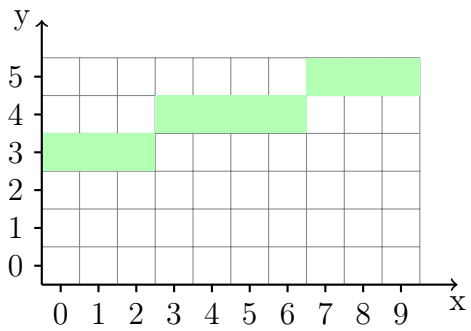

Figure 4: Screen layout with $n_x = 10$ and $n_y = 6$, showing how the pixels will be coloured for the green line, using the mid point algorithm.

4 Your Task:

You are required to implement all of the functions laid out in this section.

Task Name	Marks
1_LineGeneral	14
2_DrawLineA-F	60
3_DrawLine	75
4_Shapes	17
Testing	14

Table 1: Marks per Task

4.1 Screen

This is the file that will act as the interface between your code and the terminal. Do not change this file as it has been provided for you.

4.1.1 Functions

- `setUpScreen`
  - This function is used to define the size of the screen and takes as input the number of rows (n_x) and columns (n_y) the screen is comprised of.
- `cleanScreen`
  - This function is used to clear all the "pixel" values that are currently stored in the screen.
- `draw`
  - This function is used to provide a symbolic value for the "pixel" denoted by the passed in x and y coordinates.
  - This function also allows you to provide your own symbolic value that you would like to assign to the "pixel".
- `printScreen`
  - This function is used to print out the contents of the screen to the terminal.
- `cleanUpMemory`
  - This function is used to clean up any memory used by the screen.

4.2 Lines

This section deals with the process of drawing lines on the screen. The following general notation is used for the parameter names in this section:

- `P0` refers to Point 0 and `P1` refers to Point 1.
- `P0x` refers to the x component of Point 0 [1], and `P0y` refers to the y component of Point 0[2]. The same notation is used for Point 1.
- `x` refers to a value for x and `y` refers to a value for y.
- The `symbol` is the symbolic value that is used to draw the line.

[1] `P0x` can also be thought of as x_0.
[2] `P0y` can also be thought of as y_0.

4.2.1 General Functions

- `implicitLineEquation`
  - This function should calculate the result of the implicit line equation as provided by Equation (1).
  - Example: The following function call:

```
implicitLineEquation(0,1, 5, 8, 0.1, 1.2)
```

    should result in a value of 0.3.
  - *Hint, you basically need to substitute the values into the equation and give the result of the equation. You can ignore the* $= 0$*, just give the result of* $f(x, y)$*.*
- `min`
  - This function should return the smallest value between the two passed-in parameters.
- `max`
  - This function should return the biggest value between the two passed-in parameters.
- `absV`
  - This function is an overloaded function which works on `ints` and `floats`.
  - This function should return the absolute value of the passed-in value.
- `gradient`
  - This function should calculate the gradient between Point 0 and Point 1.
  - If the gradient is invalid, or undefined, return 0.

4.2.2 Line Functions

The set of functions in this sub-section implements the midpoint drawing algorithm. Each of the functions takes in the same parameters, which were discussed at the start of Section 4.2.

Firstly, the two special cases are discussed, namely `drawLineA` and `drawLineB`. These two functions deal with whether the line is drawn perfectly vertically or horizontally.

- `drawLineA`
  - This function is used to draw vertical lines, by iterating from Point 0 to Point 1, drawing a symbol at each iteration.
  - *Hint: vertical lines have the same* x *value but the* y *value differs*

- `drawLineB`
  - This function is used to draw horizontal lines, by iterating from Point 0 to Point 1, drawing a symbol at each iteration.
  - *Hint: horizontal lines have the same y value but the x value differs*

Hint: the `min` *and* `max` *functions can come in handy for* `drawLineA` *and* `drawLineB`*.*

Next, the more complex cases are discussed. This includes `drawLineC` all the way to `drawLineF`. Each of the functions' implementation is given below in pseudo-code format, where `F` on line 5 in the pseudocode refers to `implicitLineEquation`:

- `drawLineC`

Algorithm 1 Pseudo code for `drawLineC`

```
1: procedure DRAWLINEC(p0x, p0y, p1x, p1y, symbol)
2:     y ← p0y
3:     for x ← p0x to p1x do
4:         DRAW(x, y, symbol)
5:         if F(p0x, p0y, p1x, p1y, x + 1, y + 0.5) < 0 then
6:             y ← y + 1
7:         end if
8:     end for
9: end procedure
```

- `drawLineD`

Algorithm 2 Pseudo code for `drawLineD`

```
1: procedure DRAWLINED(p0x, p0y, p1x, p1y, symbol)
2:     y ← p0y
3:     for x ← p0x to p1x do
4:         DRAW(x, y, symbol)
5:         if F(p0x, p0y, p1x, p1y, x + 1, y - 0.5) > 0 then
6:             y ← y - 1
7:         end if
8:     end for
9: end procedure
```

- `drawLineE`

Algorithm 3 Pseudo code for `drawLineE`

```
1: procedure DRAWLINEE(p0x, p0y, p1x, p1y, symbol)
2:     x ← p0x
3:     for y ← p0y to p1y do
4:         DRAW(x, y, symbol)
5:         if F(p0x, p0y, p1x, p1y, x + 0.5, y + 1) > 0 then
6:             x ← x + 1
7:         end if
8:     end for
9: end procedure
```

- `drawLineF`

Algorithm 4 Pseudo code for `drawLineF`

```
1: procedure DRAWLINEF(p0x, p0y, p1x, p1y, symbol)
2:     x ← p0x
3:     for y ← p0y down to p1y do
4:         DRAW(x, y, symbol)
5:         if F(p0x, p0y, p1x, p1y, x + 0.5, y - 1) < 0 then
6:             x ← x + 1
7:         end if
8:     end for
9: end procedure
```

Very important, `drawLinesC` to `drawLineF`, requires that `p0x` < `p1x`. If the passed in parameters violate this restriction, you need to swap the points such that Point 0 becomes Point 1 and Point 1 becomes Point 0. *Using recursion with a change of passed in parameters, you can easily achieve this.*

Lastly, the `drawLine` function is the function that is responsible for correctly invoking `drawLineA` to `drawLineF` based on the provided input.

- `drawLine`
  - If the passed in parameters require a vertical line to be drawn, `drawLineA` should be called.
  - If the passed in parameters require a horizontal line to be drawn, `drawLineB` should be called.
  - If the passed in parameters require a line to be drawn with a gradient of $0 < m < 1$, then `drawLineC` should be called.
  - If the passed in parameters require a line to be drawn with a gradient of $-1 < m < 0$, then `drawLineD` should be called.
  - If the passed in parameters require a line to be drawn with a gradient of $m \geq 1$, then `drawLineE` should be called.
  - If the passed in parameters require a line to be drawn with a gradient of $m \leq -1$ then `drawLineF` should be called.

4.3 Shapes

Using the `drawLine` function in the `Line` file, we can draw a series of shapes. You will be drawing two shapes, namely a triangle and a square. For both of the functions in this section, the `symbol` parameter dictates the symbolic value of the line that will be drawn.

4.3.1 Functions

- `drawTriangle`
  - This function is used to draw a triangle based on the passed-in parameters.
  - This function needs to draw three lines:
    - Point 1 to Point 2
    - Point 2 to Point 3
    - Point 3 to Point 1
- `drawSquare`
  - This function is used to draw a square-like object based on the passed-in parameters.
  - This function needs to draw four lines:
    - TL (top left) to TR (top right)
    - TR to BR (bottom right)
    - BR to BL (bottom left)
    - BL to TL

5 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the assignment marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the `main.cpp` file) that will be used to test an Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the gcov [3] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j ${files}
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

We will scale this ration based on class size.

The mark you will receive for the testing coverage task is determined using Table 2:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 2: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the functions stipulated in this specification will be considered to determine your testing mark. Remember that your main will be testing the Instructor Provided code and as such, it can only be assumed that the functions outlined in this specification are defined and implemented.

As you will be receiving marks for your testing main, we will also be doing plagiarism checks on your testing main.

[3] For more information on gcov please see `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

6 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on FitchFork.
- You may only use **C++98**.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on FitchFork.
- Do not include using `namespace std` in any of the files.
- You may only use the following libraries:
  - <iostream>

7 Upload Checklist

The following C++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- `Lines.cpp`
- `Shapes.cpp`
- `main.cpp`
- Any textfiles used by your `main.cpp`

The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

8 Submission

You need to submit your source files on the FitchFork website (`https://ff.cs.up.ac.za/`). All methods need to be implemented (or at least stubbed) before submission. Your code should be able to be compiled with the following command:

```
g++ -std=C++98 -g -Wall -Werror *.cpp -o main
```

and run with the following command:

```
./main
```

Remember your `h` file will be overwritten, so ensure you do not alter the provided `h` files.
You have 20 submissions and your final submission's mark will be your final mark. Upload your archive to the Practical 7 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**

9 Examples

This section contains a series of programs with example output:

9.1 Example 1

Code:

```
    Screen::setUpScreen(10, 6);
    Screen::printScreen();
    Screen::cleanUpMemory();
```

Output:

```
+----------+
|          |
|          |
|          |
|          |
|          |
|          |
+----------+
```

9.2 Example 2

Code:

```
    Screen::setUpScreen(10, 6);
    drawLine(0,0,10,10,'@');
    Screen::printScreen();
    Screen::cleanUpMemory();
```

Output:

```
+----------+
|      @   |
|     @    |
|    @     |
|   @      |
|  @       |
| @        |
+----------+
```

9.3 Example 3

Code:

```
    Screen::setUpScreen(10, 6);
    drawLine(0,0,10,10,'@');
    drawLine(9,0,0,6,'^');
    Screen::printScreen();
    Screen::cleanUpMemory();
```

Output:

```
+----------+
| ^^  @    |
|   ^@     |
|   @^^    |
|  @   ^   |
| @     ^^ |
|@        ^|
+----------+
```

9.4 Example 4

Code:

```
    Screen::setUpScreen(10, 6);
    drawTriangle(3,3,0,0, 8,0);
    Screen::printScreen();
    Screen::cleanUpMemory();
```

Output:

```
+----------+
|          |
|          |
|   x      |
|  x xx    |
| x    xx  |
|xxxxxxxxx |
+----------+
```

9.5 Example 5

Code:

```
    Screen::setUpScreen(20, 20);
    drawSquare(0,19, 15, 15, 14, 5, 3, 3, '$');
    Screen::printScreen();
    Screen::cleanUpMemory();
```

Output:

```
+--------------------+
|$$                  |
|$ $$$$              |
|$     $$$$          |
| $        $$$$      |
| $            $$    |
| $              $   |
| $              $   |
| $              $   |
| $              $   |
|  $            $    |
|  $            $    |
|  $            $    |
|  $            $    |
|  $            $    |
|   $         $$$    |
|   $  $$$$$$        |
|   $$$              |
|                    |
|                    |
|                    |
+--------------------+
```