Programming 2 (3EC version) — Practicals Manual (v20.1)

Robotics and Mechatronics

Tuesday 2nd February, 2021

Contents

0 Introduction **5**
0.1 Context . . . 5
0.2 Organisation . . . 5
0.3 Equipment / Textbook . . . 5
0.4 Responsibility and fraud . . . 6
0.5 Quality of the results . . . 7
0.6 Document History and Disclaimer . . . 7

1 First Assignment **9**
1.1 BMI Calculator . . . 9
1.2 Heart Rates . . . 9
1.3 Submission and Questions . . . 10

2 The Maze **11**
2.1 Arrays . . . 11
2.2 Recursion . . . 11
2.3 Submission and Questions . . . 11

3 Lists and searching **13**
3.1 Linked List . . . 13
3.2 Tree . . . 13
3.3 Submission and Questions . . . 14

4 Inheritance **15**
4.1 Package Inheritance Hierarchy . . . 15
4.2 Pacman (3 EC) . . . 15
4.3 Submission and Questions . . . 15

5 Complexity **17**
5.1 Algorithm 1: Scanning . . . 17
5.2 Algorithm 2: Marching Squares . . . 17
5.3 Algorithm 2++: Improved Marching Squares . . . 18
5.4 Algorithm 1++: Multithreaded Scanning . . . 18
5.5 Submission and Questions . . . 19

6 Final Assignment (3 EC) **20**
6.1 GUI library . . . 20
6.2 Challenge 1: Creating the board . . . 20

6.3 Challenge 2: Creating Pacman . . . 21
6.4 Challenge 3: Creating dots . . . 21
6.5 Challenge 4: Creating Ghosts . . . 21
6.6 Challenge 5: Ghosts . . . 21
6.7 Challenge 6: Creating energizers . . . 22
6.8 Challenge 6: Creating fruit . . . 22
6.9 Submission . . . 22

A Change Log **23**

B Installation of C++ development environment **24**
B.1 Introduction . . . 24
B.2 Enable C++11 or newer . . . 25
B.3 Compiler settings for C++11 threads . . . 25
B.4 The test case . . . 25
B.5 Common issues . . . 26

C Standard comment header **27**

D Program directory structure **28**

E Useful resources **29**
E.1 Books . . . 29
E.2 Online . . . 29

References **30**

0 Introduction

0.1 Context

The goal of this practical work is to train yourself in software development using C++, focusing on the essentials of the language and learning good programming practices. This implies that structuring and documenting the delivered software is as important as its performance (that is, computing the correct answers reasonably fast).

To reach this goal, the exercises start as training exercises and end with a somewhat larger project.

You need to have a C++ development environment installed on your *own* laptop. See Appendix B for more detailed information on what to use and how to install it.

0.2 Organisation

The practical assignments consist of 5x8 + 1x16 hours in the class room. During this time each group (2 persons) must complete the practical assignments. At the end of each week, you must submit an assignment via the correspinding Assignment in Canvas. Each assignments is described in a separate chapter in this document. Chapter 6 describes the final project.

You can ask questions regarding the assignments to the Teaching Assistants, either face2face when you are present at the on-Campus practicals, or on line when you are at home during the practicals. Asking questions at other times is also possible. See Canvas about what communication forms we offer (page on Organisation of Programming 2).

0.3 Equipment / Textbook

You have to use your 'own' computer / laptop.

Next to this handout, we will use the textbook by Deitel and Deitel (2017). The assignments in this handout are based on the assignments in this textbook.

0.3.1 Assignments

This document contains multiple assignments, growing in creativity/freedom and complexity. The subjects of the assignments are based on the content of the lectures.

Next to the programming assignments, the exercises also have questions to be answered as part of each exercise.

0.3.2 Planning

Week	*Practicals*	*Subject*	*Deitel & Deitel*	*Submission deadline*
5	4 Feb, 5 Feb	Installation & Get Familiar	Chap. 1, 2, 3	5 Feb
6	11 Feb, 12 Feb	Arrays, Pointers, Recursion	Chap. 6, 7, 8	12 Feb
7	18 Feb, 19 Feb	Lists & Trees	Chap 19	19 Feb
8		*Holidays*		
9	3 Mar, 5 Mar	Inheritance	Chap. 9, 11, 12	5 Mar
10*	11 Mar, 15 Mar	Sorting/Multi-threading	Chap. 20	15 Mar
11	16 Mar, 18 Mar	Final Assignment	-	-
12	25 Mar, 26 Mar	Final Assignment	-	26 Mar

Submission deadline is the very late evening of the last half day of each practical: usually a Friday of that week, effectively Saturday 04:00.

Note the following:

- Not all practicals are scheduled to be on Campus.
- During the practicals, the TAs are available to answer questions, both virtual and face2face (when practical is scheduled on Campus.
- Presence on Campus when the practicals are scheduled on Campus is not necessary. Obviously, being there makes it easier to ask questions and get answers.

0.3.3 Recording Intermediate Progress

Show your work to one of the TAs (Teaching Assistants) before the end of the last practical halfday of each exercise in case you are on Campus. You also have to submit your work (sources etc, and answers to questions) via Canvas. Be sure that the TAs can easily compile and run your code on their machines. The submission deadline for each assignment is at the very end of the evening of the last day scheduled for each assignment (usually 04:00 Saturday).

Each submission will be graded. The grade may also be indicated by a traffic-light colour: Red means insufficient, Amber (Yellow) means borderline: you should put more effort into your work, and Green means it is sufficient.

At the end of the course, the mean of all these marks, together with a grade given by the TAs and the teacher, determines your final grade. All assignments are weighed the same, except for the final assignment which weighs twice as much as it is twice as large.

0.4 Responsibility and fraud

All group members share responsibility for the work handed in. When you distribute work among group members, check each other's work, discuss the work among all group members, do you understand it all?

You must submit original work.

Plagiarism, i.e. copying of someone else's work without proper reference, is a serious offense which in all cases will be reported to the Examination Board. Refer to UT's Student Charter Student Charter (2020).

In cases where a non-trivial amount of work is copied *with* proper reference, indicate which parts are copied and which parts are your own original work. The copied work will not be considered for grading.

This also holds for the assignments where code is explicitly provided (e.g. Assignments 3 and 6): only your extensions are considered.

0.4.1 What constitutes fraud?

When it comes down to handing in assignments, every year there are students who do not understand the borderline between, on the one hand, cooperating and discussing solutions between groups (which is allowed), and on the other, copying or sharing solutions (which is forbidden and counted as fraudulent behaviour). Here are some scenarios which may help in making this distinction.

- **Scenario.** Peter and Lisa are quite comfortable with programming and have pretty much finished the assignment. Mark and Wouter, on the other hand, are struggling and ask Lisa how she has solved it. Lisa, a friendly girl, shows her solution and takes them through it line by line. Mark and Wouter think they now understand and go off to create their own solution, based on what they saw. Is this allowed or not?
  Verdict. No problem here, everything is in the green. It is perfectly fine and allowed for

Lisa to explain her solution, even very thoroughly. The important point is that in implementing it themselves and testing their own solution, Mark and Wouter are still forced to think about what is happening and will gain the required understanding, though probably they will not get as much out of it as Lisa (explaining stuff to others is about the best possible way to learn it better yourself!).

- **Scenario.** The start is as in the previous case. However, while Mark and Wouter implement their own solution, inspired by that of Lisa, some error crops up which they do not understand. Lisa has left by now; after they mail her, still trying to be helpful she sends them her solution for them to inspect. They inspect it so closely that in the end their solution is indistinguishable from Peter and Lisa's, except for the choice of some variable names and the comments they added themselves. Is this allowed or not?
  Verdict. This is now a case of fraud. Three are at fault: Lisa for enabling fraud by sending her files (even if it was meant as a friendly gesture) and Mark and Wouter for copying the code. Peter was not involved, developed his own solution (together with Lisa) and is innocent.

- **Scenario.** Alexandra and Nahuel are not finished, and the deadline is very close. The same holds for Simon and Jaco. On the Friday night train home, Jaco and Nahuel meet and during the 2-hour train ride work it out together. They type in the same solution and hand it in on behalf of their groups. Is this allowed or not?
  Verdict. This is also a case of fraud. Actually there are two problems here. The first is that both Nahuel and Jaco handed in code on behalf of their groups that had been developed by them alone, without their partners. This is unwise and against the spirit of the assignment (Alexandra and Simon also need to master this stuff!) but essentially undetectable and not fraudulent. The second problem is that the solution was developed, and shared, in collaboration between two groups; this is definitely forbidden. All four students are culpable; Alexandra and Simon cannot hide behind the fact that they did not partake in the collaboration, as they were apparently happy enough to have their name on the solutions and pretend they worked on it, too.

Note that we are not on a witch-hunt here: let us stress again that cooperating and discussing assignments is OK, even encouraged; it is at the point where you start copying or duplicating pieces of code that you cross the border.

0.5 Quality of the results

The quality of the code will be judged, implying:

- The code must compute the correct values in reasonable time.

- The structure of the code must be clear, and easily understandable by the TAs.
  You can use *pseudo code* to structure your design, see Deitel and Deitel (2017, Sec 4.3).

- Names of identifiers must follow an appropriate naming scheme.
  Follow the examples as given in Deitel and Deitel (2017).

- The code must contain effective comments.
  Follow the style as shown in Deitel and Deitel (2017), and explained in their Chapter 2.
  No comments or irrelevant comments lead to an insufficient mark.

See also tips and hints on the slides dealing with programming style and design decisions.

0.6 Document History and Disclaimer

When you find a mistake or have remarks about this document, please contact one of the TAs.

As we also continuously improve this document, newer versions appear regularly, see its date and version number. Changes are summarized in the Changelog, in Appendix A. The latest version can be found on Canvas. Be sure to always use the latest version of this guide.

This document has seen many versions over the years. Contributors to this document: Wolfgang Baumgartner, Jan Broenink, Johan Engelen, Sander Grimm, Silke Hofstra, Karim Kok, Frits Kuipers, Laurie Overbeek, Chris Zeinstra, Gijs van Oort.

Special thanks to Arend Rensink for the subsection “What constitutes fraud?”.

1 First Assignment

At the start of every file you create, make sure you use the standard comment header, see Appendix C.

1.1 BMI Calculator

To determine whether a person is overweight or obese, you can use a measure called the Body Mass Index (BMI). The formula for calculating the BMI is:

$$\text{BMI} = \frac{\text{weight in kilograms}}{(\text{height in meters})^2} \tag{1.1}$$

(a) Create a BMI calculator application that reads the user's weight in kilograms and height in meters, and then calculates and displays the user's body mass index. Structure your program well, use functions!

(b) Create a function called `evaluateAndPrintBMI` that displays the category of the calculated BMI according to the table below. For example, when the calculated BMI is 21, the application should say something like "Your weight is normal.".

(c) The application should display the following information from the Department of Health and Human Services/National Institutes of Health so the user can evaluate his/her BMI. You should create a separate function called `printInfo` for this.

BMI VALUES:	
Underweight:	less than 18.5
Normal:	between 18.5 and 24.9
Overweight:	between 25 and 29.9
Obese:	30 or greater

1.2 Heart Rates

While exercising, you can use a heart-rate monitor to check that your heart rate stays within a safe range suggested by your trainers and doctors. According to the American Heart Association (AHA), the formula for calculating your maximum heart rate in beats per minute is 220 minus your age in years. Your target heart rate is a range that is 50-85% of that age-specific maximum.

(a) Create a class to store information about humans: the class attributes should include the person's first name, last name and date of birth (consisting of separate attributes for the month, day and year of birth).

(b) The class should have a constructor that receives this data (first name, last name, birthday, birth month, birth year) as parameters. Provide `set` and `get` functions where they make sense (for example, it's nice to be able to get a person's first name, but it does not make much sense to set a person's first name separate from the last name). The set function for the date of birth should check that the day (1-31, let's simplify) and month (1-12) are in the correct range.

(c) The class also should include a function `getAge` that calculates and returns the person's age (in years). If you do not know how to obtain the current date from the computer, you should prompt the user to enter the current month, day and year before calculating the person's age.

(d) Create a function `calculateMaximumHeartRate` that calculates and returns the person's maximum heart rate and a function `calculateTargetHeartRates` that calculates and returns the person's minimum and maximum target heart rate.

(e) Write an application that prompts for the person's information and uses your class for human info (instantiate an object). Next, the application should show the full name, date of birth, the person's age (years), maximum heart rate, and target heart rate range. Test your program; for example, test that it validates the dates entered by the user.

1.3 Submission and Questions

After you are done with both exercises, show your work to one of the TAs.

Hand in your work via Canvas with the two projects above in a zip called [Group number]_3EC_Assignment1.zip (eg. `01_3EC_Assignment1.zip` for Group 1). Don't add the object files (*.o, *.obj) or executable files to the zip file; only the *.cpp and *.h files is enough. Next to this zip file hand in a single A4 page *PDF file* with explanations of the design choices you made in the exercises and the answers to the following questions:

(a) Why did you have to include `iostream` in your code?

(b) What is the difference between `private` and `public` access specifiers in a C++ class?

(c) Explain the purpose of data hiding.

(d) How can a program use the standard library class `string` without using a `using namespace` directive? What is problematic about putting a `using namespace` directive at the start of your code?

(e) Give the output of the following code and explain.

```
#include <iostream>

void function();

int a = 20;

int main() {
    int a = 10;
    function();
    std::cout << a;
    return 0;
}

void function() {
    std::cout << a << std::endl;
}
```

2 The Maze

The grid of hashes and dots in Figure 2.1 is a two-dimensional 12-by-12 array representation of a maze. In this two-dimensional array, the hashes represent the walls of the maze and the dots represent squares in the possible paths through the maze. The `x` marks the maze entrance. Moves can be made only to a location in the built-in array that contains a dot.

There is a simple algorithm for walking through a maze that guarantees finding the exit (assuming there is an exit). If there is no exit, you'll arrive at the starting location again. Place your right hand on the wall. If the maze turns to the right, you follow the wall to the right. As long as you do not remove your hand from the wall, eventually you'll arrive at the exit of the maze. There might be a shorter path than the one you've taken, but you are guaranteed to get out of the maze if you follow this algorithm.

```
# # # # # # # # # # # #
# . . . # . . . . . . #
. . # . # . # # # # . #
# # # . # . . . . # . #
# . . . . # # # . # . x
# # # # . # . # . # . #
# . . # . # . # . # . #
# # . # . # . # . # . #
# . . . . . . . . # . #
# # # # # # . # # # . #
# . . . . . . . . . . #
# # # # # # # # # # # #
```

Figure 2.1: Two-dimensional array representation of maze.

2.1 Arrays

(a) To store the maze fields you can use a two dimensional-array using `std::array` from the standard library. To use `std::array` you need to include `<array>`. To see how a such an array is made look at `http://en.cppreference.com/w/cpp/container/array`. Initialize a maze with the maze in Figure 2.1.

(b) Make a function that prints all the elements of the maze, just as in Figure 2.1.

2.2 Recursion

(a) Make a function that determines the start position of the maze by searching the array for the starting 'x'. You can use this function later to start your recursive traversal function (next part).

(b) Write a recursive function `traverseMaze` to walk through the maze. As `traverseMaze` attempts to locate the exit from the maze, it should place the character 'X' in each square of the path. The function should display the maze after each move, so the user can watch as the maze is solved. When the maze is solved the program should print a text that the maze is solved and exit the program.

2.3 Submission and Questions

After you are done with both exercises, show your work to one of the TAs.

As in the previous assignment, hand in your workspace with the two projects above in a zip called [Group number]_3EC_Assignment2.zip (eg. `01_3EC_Assignment2.zip` for Group 1). Next to this zip file hand in a single-page PDF file with explanations of the choices you made in the exercises (include this file in the zip file). This document should explain how your maze traversal solution works. Also include the answers to the following questions:

(a) Explain in your own words what recursion is.

(b) Recursion is not really an appropriate mechanism for the right-hand-on-the-wall solving algorithm. Why not? How about using recursion for finding the shortest path?

(c) Create a range-based loop using the `auto` keyword to loop trough the values of array `a` of type `std::array`.

(d) When would you use the `const` keyword?

(e) What is the advantage of the `std::array` type in comparison to built-in arrays (e.g. `int a[]`)?

(f) Find the error, and explain why it is wrong:

(1)

```
#include "array"
```

(2)

```
if (a = 1) {
    b = 10;
}
```

(3)

```
int b[10] = {};
for (int i = 0; i <= 10; ++i)
    b[i] = 1;
```

(4)

```
const int arraySize = 20;
// ...
arraySize = 10;
```

3 Lists and searching

Download the example code of the textbook via the following link:

http://media.pearsoncmg.com/ph/esm/deitel/cpp_htp_9/code_examples/code_examples.html

The code of chapter 19 is used in this exercise.

Note that in this assignment, the List and Tree code is in header files and not in separate implementation (.cpp) files. If you have time left after finishing the assignment, try to find the answer to why this is necessary (hint: templates and separate compilation).

3.1 Linked List

(a) Start from the LinkedList starterkit provided on Canvas. Use the `main.cpp` file and include the header file needed to use the linked list `List` class. Instantiate two empty linked lists (the type of the list elements should be `char`).

(b) Create a function `fillList` which takes a string and a char list as argument and fills that list with the chars from the string.

(c) Use `fillList` to fill the two lists instantiated in the main function with two strings "singlylinkedlist" and "abcdefg". Print the two lists to the screen using their `print` method.

(d) Extend the `List` class (so edit `List.h`) with a function `concatenate` which takes a `List` as argument. This function should concatenate the list in the argument to the current list. The argument list needs to be emptied. You may not use the `insert*` and `remove*` functions from the `List` class.

(e) In your main function, create two new lists and fill them with "hijklmnop" and "qrstuvw". Use your `concatenate` method to append these two lists to the list that contains "abcdefg". Print the appended list.

3.2 Tree

(a) Start from the Tree starterkit provided on Canvas. Use the `main.cpp` file. Instantiate a tree of integers and fill it with 10 different predefined integers (unsorted!).

(b) Create a `search` function in the `Tree` class that attempts to locate a specified value in a binary search tree object. The `search` function should take the search key to locate as argument. You can choose between a recursive search implementation (create a search helper function) or an iterative search implementation. If the node containing the search key is found, the function should return a pointer to that node; otherwise, the function should return `nullptr`.

(c) Extend your main program such that it asks the user to input a number. The program should search your predefined array for the user's number using the `search` function created previously. The program should output whether the number was found or not.

(d) Create a member function `outputTree` to display a binary tree object on the screen. The function should output the tree row by row, with the top of the tree at the left of the screen and the bottom of the tree toward the right end of the screen. Each row is output vertically. For example, the binary tree illustrated in Figure 3.1. Use a recursive algorithm for this.

```
                50
          48
                47
     42
          33
27
                20
          17
     13
                11
          6
                1
```

Figure 3.1: Example output of outputTree function.

3.3 Submission and Questions

After you are done with the exercises, show your work to one of the TAs and hand in a zip file named "[Group number]_3EC_Assignment3.zip" as before. Next to the zip file hand in a PDF file explaining your design choices and include answers to the questions below.

(a) What are the differences between a Linked List, a Stack, a Queue, and a Tree?

(b) Give example usages for these four different data structures.

(c) How do you convert a Singly Linked List to a Doubly Linked List and to a Circular Linked List?

(d) When do you use the `&` sign and when do you use the `*` sign when using pointers?

(e) You define a new pointer variable, but you don't have an address yet to initialize the pointer with. With what value should you initialize that pointer and why?

(f) At the end of (or during) your Linked List program, the Lists' nodes are destroyed and the program prints the destroyed node elements. Because you instantiated more than one list, you see the order in which the lists are destroyed. Explain this order: what determines the order in which the lists are destroyed, is the order always the same? Note: this is about the destruction of the List objects themselves, not their elements.

4 Inheritance

4.1 Package Inheritance Hierarchy

Package-delivery services, such as DHL® and UPS®, offer a number of different shipping options, each with specific associated costs and other attributes. For example, there are the `TwoDayPackage` and `OvernightPackage` package options. The cost of shipping a package depend on the type of package. The cost of a `TwoDayPackage` is 2.50 euro per kg, with a flat fee of 5 euro. The cost of an `OvernightPackage` is equal to a `TwoDayPackage` but with the additional cost of the weight squared times 1.10 euro/kg^2.

Your task is to write an object-oriented program that creates a few packages with different shipping options (can be hard-coded). This set of packages should then be passed to a function `printCosts` that loops over this set of packages and prints the cost of shipping for each package. The cost for each package must obtained by calling the package's member function `calculateCost`.

- Create a class hierarchy (inheritance) to represent the various types of packages. Use `Package` as the base class name.
- At least the following information should be stored for each package: name, address, city, state and ZIP code for both the sender and the recipient of the package; and the package's weight.
- Note that the function `printCosts` does not need to know about the different types of packages. If a new package type is added, the `printCosts` function should not need to be modified at all!

4.2 Pacman (3 EC)

Make class diagrams of the classic game Pacman and explain your decisions. Show the relations between the different classes. Include all important attributes and functions. See:

`http://web.archive.org/web/20200309081227/http://www.pacmangame.org/classic-pacman/`
(the original, `http://www.pacmangame.org/classic-pacman/`, is offline now)

You can draw the class diagram in whichever program you prefer. Easy-to-use software for class diagrams (UML diagrams) are the Draw.io website `www.draw.io` and Dia (`https://wiki.gnome.org/Apps/Dia`).

Put your Pacman class diagram and design choices in its own document (separate from the design choices of the “Package Inheritance Hierarchy" part).

4.3 Submission and Questions

After you are done with the exercises, show your work to one of the TAs and hand in a zip file named "[Group number]_3EC_Assignment4.zip" as before. Next to the zip file hand in a PDF file explaining your design choices and include answers to the questions below.

(a) Explain why inheritance is used in this assignment.

(b) In the first assignment you have explained the difference between private and public access specifiers. What is the difference between them and the protected access specifier?

(c) Why should you be careful when using multiple inheritance?

(d) Explain composition and inheritance in common words, using the topic "computer". (as a start: what classes/objects are related to a computer?)

(e) The compiler "inserts" calls to destructors in your code. Where are these calls to destructors inserted? (in other words, when are they called during program execution?)

(f) What is a virtual destructor, when is it necessary and why?

5 Complexity

Consider a scalar potential field P with two potential peaks a and b:

$$P(x,y) \mapsto \frac{1}{(x-a_x)^2+(y-a_y)^2} + \frac{1}{(x-b_x)^2+(y-b_y)^2} \tag{5.1}$$

We define $x = 0$ and $y = 0$ to be the middle of the screen, and now want to display where the potential has a certain value v.

In this assignment, we'll look at different ways of doing just that. You have to implement two algorithms: "Scanning" and "Marching Squares".

There is a starter kit available on Canvas that you must use. The starter kit sets up the display, defines the potential function, and provides the logic to use an algorithm to draw a contour and switch between different algorithms. Using the starter kit, you only have to implement the algorithms themselves that determine which pixels light up.

Practical note: to best see the differences in performance of the algorithms, it may be best to turn off the compiler's optimizer (`/Od` or `-O0`).

5.1 Algorithm 1: Scanning

The "Scanning" algorithm is fairly simple. We scan the whole display area from top-left to bottom-right, horizontally or vertically. For each pixel: we calculate $P(x,y)$ and see whether the local value is above or below the value v (thus, v can be seen as a threshold value). If the outcome (above or below) is different from the previous pixel, we know we have crossed the threshold and we draw the pixel white. If there is no difference from the previous pixel, we don't draw anything (black).

As you will see, the result displayed is not very satisfactory and the algorithm will thus need further improvement but we won't work on that in this course.

5.2 Algorithm 2: Marching Squares

The algorithm in this section is a 2D variant of the famous Marching Cubes algorithm of Lorensen and Cline (1987).

The "Scanning" algorithm is not fast enough to result in a smooth animation. In general though, there is no faster way: we have to inspect each pixel to know whether the potential field has value v there (or close to it).

However, the potential field in this assignment has a lot more structure and we can exploit that. For example, we know that P is continuous, which means that $P(x,y) = v$ results in continous closed curves. You can use all other knowledge you have about the potential field of this assignment.

The fact that the pixels to be drawn are on a continuous curve helps a lot: once we have found one pixel on the curve, we only have to look at its neighbours to trace the rest of the curve. And thus we will not have to scan the whole area any more. This insight leads to the "Marching Squares" algorithm:

1. Find one pixel on the curve (e.g. by scanning). Add that pixel to a `worklist`.
2. While `worklist` is not empty do:
   (a) Take and remove pixel p from the worklist.

(b) Add p to the `visitedlist`. If p was already in `visitedlist`, skip to the next iteration of the loop.

(c) Calculate P at the four corners of the pixel, $(x-\frac{1}{2}, y-\frac{1}{2}), (x-\frac{1}{2}, y+\frac{1}{2}), (x+\frac{1}{2}, y-\frac{1}{2}), (x+\frac{1}{2}, y+\frac{1}{2})$, and check if the $P = v$ curve crosses the line segments between these corners. In other words, check whether the P values at all corners are either above or below v. If the $P = v$ curve crosses one of the segments: draw the pixel!

(d) If we had to draw the pixel, add all 8 neighbours of the pixel to the `worklist`.

Note that `worklist` and `visitedlist` do *not* need to be 'list' data structures! Knowing what operations are needed, what data structures should be used for them? (the `worklist` datastructure and the `visitedlist` data structure will differ)

5.3 Algorithm 2++: Improved Marching Squares

After implementing the "Marching Squares" algorithm, you'll notice that there is something wrong: it does not take into account that there may be more than one curve to be drawn! Also, the algorithm is not much faster than the scanning algorithm...

Some hints to improve your algorithm:

1. What is the complexity of the two phases of the algorithm: (1) finding the first pixel on the contour and (2) marching?
2. It is perfectly fine to add more than one pixel to the worklist in phase 1 to kick-start the marching phase.
3. The number of curves is less or equal to the number of potential peaks (a and b in this case).

5.4 Algorithm 1++: Multithreaded Scanning

The C++ standard library contains a `std::thread` class. Threads allow functions to execute concurrently. With threads you can, possibly, improve the performance of your program. A thread can be started easily by making a `std::thread` object. The `join` function of the `std::thread` class can be used to synchronise threads. This function returns when the thread is finished. The following code example shows the creation of one new thread.

```
1   #include <thread>
2
3   void threadFunction(int x)
4   {
5       // This function runs in a separate thread!
6   }
7
8   int main()
9   {
10      // Start a new thread that calls "threadFunction"
11      std::thread t(threadFunction, 1);
12
13      // Waits for the thread to finish
14      t.join();
15      return 0;
16  }
```

Add multithreading to your implementation of the scanning algorithm using `std::thread`.

When two threads try to write and read the same memory concurrently, the behaviour of your program can be undefined. This is called a *race condition*. Race conditions can be prevented by defining critical sections in your code by using mutual exclusion (see `std::mutex`, `std::lock_guard`, `std::mutex::lock()`, `std::mutex::unlock()`). Keep this in mind when using shared data in threads. Synchronization is not needed if, during the multithreaded part, the program is only reading the shared data. Drawing pixels onto the screen (SDL2 function call) involves writing data, and thus synchronization is needed for multithreaded pixel drawing.

Think about the amount of threads that is likely to improve performance. Creating and stopping a thread requires some overhead time, and your CPU only has finite resources. Think about whether you need critical sections (synchronization using a mutex), and make them as small as possible. Otherwise other threads will be waiting on the one thread that locked the mutex...

Some compilers need to know your program will use threads, see Appendix B for the correct settings.

5.5 Submission and Questions

After you are done with the exercises, show your work to one of the TAs and hand in a zip file named "[Group number]_3EC_Assignment5.zip" as before. Next to the zip file hand in a PDF file explaining your design choices and include answers to the questions below.

(a) What was the complexity of your function in the Maze assignment that determines the start position of the maze by searching the maze array for the starting 'x'? Use n equal to the maze width (i.e. the maze is $n \times n$).

(b) What data structures did you use (array, vector, list, tree, ...), and how did the complexity of the operations influence your design decisions? (For maximum points, the answer to this question should be part of your design decisions, and here you just refer to that...)

(c) Explain the order of complexity for the scanning algorithm and the marching squares algorithm, in terms of N for a screen size of $N \times N$ pixels.

6 Final Assignment (3 EC)

In the final assignment you will make a Pacman game! The class diagram created in the previous exercise should be implemented.

This is a large and challenging exercise. We expect you to finish at least the first 4 challenges for a sufficient mark.

Show your work after each day to one of the TA's.

6.1 GUI library

A simple GUI Pacman starterkit was written for this exercise. This starterkit provided on Canvas (`pacmanLib.zip`) uses the SDL2 starterkit https://www.libsdl.org/. Follow the `README.md` file to install SDL2 and to get the starterkit to compile and run.

The starterkit uses a 2D `std::vector` for the map (similar to the recursion exercise). The starterkit uses sprites from a sprite sheet. A sprite is a two-dimensional image that is integrated into a larger scene.

The starterkit consists of three folders:

`resources`	contains the sprite sheet.
`src`	contains the C++ source including a skeleton main file.
`include`	contains C++ header files.

The source folder contains a skeleton main file you can use to create the pacman game. The main file contains a loop that runs constantly to render new frames to the screen. Input events are also handled in this loop in a case structure. Setting the score and the amount of lives left is also done in this loop by two the two functions `setLives` and `setScore`. These functions change the values shown on the screen.

The UI uses a `std::vector` containing the type `GameObjectStruct`. This vector contains all the objects that are rendered to the screen. The type `GameObjectStruct` contains the position and type and direction of the objects. The possible different types are listed in the `GameObjectStruct` header file. The different directions are also listed in the file as a `enum` type.

The given main file also starts a timer with a *callback* function. The callback function is `gameUpdate(Uint32, void *)`. This callback function is called every 100ms. It can be used to realise game mechanics like the movement speed of Pacman.

6.2 Challenge 1: Creating the board

In this challenge the board will be created for the Pacman game. In the previous exercise a class diagram of the game Pacman is made. Create a new project for the Pacman game and create the class structure you have determined before. You don't need to implement these classes yet. The only classes that will be implemented in this challenge are the classes which are responsible for creating the game board and initialising the game. Use the provided GUI library to display the game board on the screen with the walls. Also the amount of points and lives must be visible. The layout of the board is also provided in the file 'board.h'. In the provided 'main.cpp' file there is a function where you can give the layout of the map. This function reads all the 1's for a wall and creates an image of the board.

Requirements

- Create a new C++ project for the Pacman game.

- Create the class structure in the C++ project. (Use the class wizard of eclipse)
- Implement the classes which are responsible for creating the pacman game board and initialising the game.
- The walls must be visible on the board.
- The score must be visible on the board
- The amount of lives must be visible on the board

6.3 Challenge 2: Creating Pacman

In this challenge Pacman will be included in the game. Pacman should be able to move in response to the users inputs. When Pacman has a moving direction, it should keep moving in that direction over time.

Requirements

- Include Pacman on the game board.
- Implement the movements of Pacman.
- Implement collisions with the wall.
- The sprite used should point in the correct direction.
- The portal function of the map should also work: When Pacman exits the map on the left side it should re-appear on the right side of the map and vice versa.

6.4 Challenge 3: Creating dots

In this challenge the dots will be included in the game. The dots represent the 'food' of Pacman.

Requirements

- Place dots on the board.
- Implement collisions with Pacman.
- Dot should be removed after collision and points should be increased.

6.5 Challenge 4: Creating Ghosts

In this challenge the ghosts will be included in the game.

Requirements

- Initialise the game with 4 ghosts in the center of the map.
- There should be 4 different ghosts (Inky, Pinky, Blinky and Clyde).
- Implement some kind of random movement for the ghosts. Ghosts should also not be able to move through walls. They may behave in the same way.

6.6 Challenge 5: Ghosts

Implement the collision behaviour of the ghosts.

Requirements

- Implement collisions between the ghosts and Pacman. On collision the position of Pacman and the ghosts should be reset to the start position. Also, the amount of lives should be decremented.

6.7 Challenge 6: Creating energizers

In this challenge the energizers will be included in the game. When Pacman eats an energiser, the ghosts become scared. Pacman is now able to eat the ghosts. When a timeout expires the ghosts stop being scared.

Requirements

- Place 4 energisers in each corner of the board.
- Implement collisions between Pacman and the energiser. On collision the ghosts should become scared, edible and a timer should be started. When this timout expires the ghosts should be normal again.
- Implement collision between pacman and a scared ghosts. On collision the ghost should reset to the start position and not be scared anymore. The amount of points should also be incremented when a ghost is eaten.

6.8 Challenge 6: Creating fruit

In this challenge the fruit will be included in the game. When Pacman eats the fruit, the points are increased by a certain amount

Requirements

- Place fruit randomly on the board after a certain amount of points gathered.
- Implement te collision between Pacman and the fruit. Increase the points on collision.

6.9 Submission

After you are finished or the time is up, show your work to one of the TAs.

Furthermore, hand in your workspace with the projects above in a zip called [Group number]_3EC_Assignment6.zip (eg. 01-3ECAssignment6.zip).

Also hand in a separate report in pdf with explanation of the choices you made in this project. Also hand in the answers to the following questions:

(a) Does the structure of classes of your implementation of the PacMan game still represent the class diagram you handed in previously? Draw a class diagram of your final implementation. Indicate the changes and motivate why you have changed it.

A Change Log

- **version 2020.1**, January 2021 / Februari 2021
  - First version Programming 2, both 3EC and 5EC versions
  - Repaired error in planning table; adapted 5EC version planning table

B Installation of C++ development environment

B.1 Introduction

As always, the installation of the tools is straightforward but you need to be very precise.

We recommend you to use the 'system' C++ compiler. Because you have to hand-in portable C++ software (meaning that it will compile and run on any of the OSses used), any compiler fully compliant with C++14 can be used. However if you want support of the TAs you will need to use software that they also know.

Appendix D provides guidance on what folder structure to use for your work.

Further instructions about how to set up your environment for the final assignment are in the `README.md` of the pacmanLib starterkit.

B.1.1 Windows

Download and install Visual Studio Community (it's free).

https://www.visualstudio.com/vs/community/

During installation, select for "workload" the "Desktop development with C++"; that should install all needed components.

Create your project as an 'empty project' and check the common issues in Section B.5.

Visual Studio's built-in debugger is great: try it out!

Have a look at "The Cherno" YouTube channel (Appendix E) for some excellent Visual Studio and C++ guidance.

B.1.2 macOS

Start the App Store and install Xcode (it's free).

Start Xcode after the installation has finished. Xcode asks if you want to install the developer tools, install them. In the terminal, run the following command:

```
xcode-select --install
```

and check that the following commands work:

```
clang++ --version
g++ --version
```

XCode has a built-in debugger.

Note: For the Networks part a copy of Eclipse can be used, with its own GUI library, as that is different from the GUI library used for Programming 2. If you choose to use Eclipse for Programming 2, it is best to have a separate Eclipse installation for the network part of the course.

B.1.3 Ubuntu (Linux)

Most Linux distributions already have the GNU GCC compiler installed. Check in the terminal that the following command works:

```
g++ --version
```

For debugging, you can use the famous commandline tool `gdb`.

Note: For the Networks part a copy of Eclipse can be used, with its own GUI library, as that is different from the GUI library used for Programming 2. If you choose to use Eclipse for Programming 2, it is best to have a separate Eclipse installation for the network part of the course.

B.2 Enable C++11 or newer

You have to make sure that the C++11 dialect of the language standard is chosen, or a newer dialect (C++14 is recommended).

Windows: Recent versions of Visual Studio default to C++14, excellent!

macOS: XCode may default to C++14, but on the commandline you have to specify `-std=c++14`.

Ubuntu, Linux: GCC may or may not default to C++14, probably on the commandline you have to specify `-std=c++14` (requires GCC >= 5).

B.3 Compiler settings for C++11 threads

Windows: There are no extra settings needed to use threads with Visual Studio.

macOS: There are no extra settings needed to use threads with the macOS compiler (clang).

Ubuntu, Linux: On Linux, you probably need to add `-pthread -lpthread` to the compiler command line.

B.4 The test case

After installing and setting up your development environment, you should be able to compile and run this example code and see an output of "The answer is 42".

```
#include <iostream>
#include <thread>

int globalVar = 1;

void threadFunction(int x)
{
    globalVar = x;
}

int main()
{
    // Start a new thread that calls "threadFunction"
    std::thread t(threadFunction, 42);

    // Waits for the thread to finish
    t.join();

    std::cout << "The answer is "
              << globalVar
              << std::endl;
    return 0;
}
```

B.5 Common issues

B.5.1 Windows/Visual Studio

The build completes, but the program disappears

Configure Visual Studio to use *console* output: Open the project properties and navigate to Linker -> System, set "SubSystem" to `Console (/SUBSYSTEM:CONSOLE)`. Run using *Start Without Debugging* (ctrl+F5).

Alternatively, set a break point at the end of the program and run using *Start Debugging* (F5).

I have included `stdafx.h` or Visual Studio asks me to

Change the following project properties:

- Linker -> System: set "SubSystem" to `Console (/SUBSYSTEM:CONSOLE)`.
- C/C++ -> Precompiled Headers: set "Precompiled Header" to "Not Using Precompiled Headers".

The `compile_and_run.bat` file doesn't work

Open the *x64 Native Tools Command Prompt for VS 2017* (or equivalent), navigate to your project directory (using `cd`) and run the script, eg:

```
cd Study\Projects\Prog2EE\assignmentX
.\compile_and_run.bat
```

C Standard comment header

For the assignments it is important that each source file contains a header with the group number and the names of the group members. Use the following header:

```
//===================================================================
// Filename    :
// Authors     :
// Version     :
// License     :
// Description :
//===================================================================
```

D Program directory structure

Common advice is to write your code as a library. When writing a library, it is convenient and common to split the source code into separate directories for *header files* and *implementation files*. The header files (extension `.h`) contain the declarations that are implemented in the implementation files (`.cpp`). When shipping your library to the users, they only need the header files, and the compiled library of object code (for example a `.a` file).

The practical assignments are too small to make this split, but for the final assignment, create the `src` and `include` folders in your project and use this structure (the provided starterkit already has that structure).

E Useful resources

E.1 Books

There are many bad C++ books, in the sense of for example containing incorrect information or teaching very bad programming style and design. Use this list to filter: https://stackoverflow.com/questions/388242/the-definitive-c-book-guide-and-list

E.2 Online

- C++ language FAQ: https://isocpp.org/faq
- C++ standard library: http://www.cplusplus.com/reference
- C++ tutorial: http://www.cplusplus.com/doc/tutorial
- C++ Core Guidelines: https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines
- "The Cherno" on Youtube: https://www.youtube.com/watch?v=18c3MTX0PK0&list=PLlrATfBNZ98dudnM48yfGUldqGD0S4FFb

References

Deitel P J, Deitel H M, 2017 *C++ How to Program*, Prentice Hall, 10th edition ISBN 978-1-292-15334-6, or 978-0-134-44823-7

Lorensen W E, Cline H E, 1987 "Marching cubes: A high resolution 3d surface construction algorithm" *SIGGRAPH Comput. Graph.* **21**, pp. 163–169, doi:10.1145/37402.37422 URL http://doi.acm.org/10.1145/37402.37422

Student Charter, 2020 "Student Charter at University of Twente" URL https://www.utwente.nl/en/ces/sacc/regulations/charter