UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science
COS132 - Imperative Programming
Practical 4

1 Introduction

Deadline: 8th of March, 20:00

1.1 Objectives and Outcomes

In constructing programs, it is necessary to provide the means to create responsive and reactive programs that might respond to changing situations. In this respect, control structures like the if statement,its companion else and the while statement are the first steps towards creating responsive and dynamic programs that are not simply linear in nature.

This practical will consist of 2 Activities and you will be required to complete all of them as part of this practical. You are also advised to consult the Practical 1 specification for information on aspects of extracting and creating archives as well as compilation if you need it. Also consult the provided material if you require additional clarity on any of the topics covered in this practical.

1.2 Structure of the Practical

Each Activity is self contained and all of the code you will require to complete it will be provided on the appropriate Activity slot.

Each Activity will require you submit a separate archive to an individual upload slot. That is, each separate Activity will require its own answer archive upload. You will upload Activity 1 to Practical4_Activity1 and so on.

1.3 Submission

Submit your code to Fitchfork before the closing time. Students are **strongly advised** to submit well before the deadline as **no late submissions will be accepted**.

1.4 Plagiarism

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes

copying someone elses work without consent, copying a friends work (even with consent) and copying textual material from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to **http://www.ais.up.ac.za/plagiarism/index.htm** (from the main page of the University of Pretoria site, follow the *Library* quick link, and then click the *Plagiarism* link). If you have questions regarding this, please ask one of the lecturers, to avoid any misunderstanding.

1.5 Mark Distribution

Activity	**Mark**
While Loops	5
Complex Operations	15
Total	**20**

2 Makefiles

Up until now, you have been receiving makefiles for your activities. However, constructing a makefile is one of the fundamental skills required for this course, and for others. Makefiles can be used in a wide variety of languages, but for the purpose of this course, makefile construction only concerns makefiles for C++ programs. This practical will provide you with an outline for constructing your own makefiles.

A makefile in C++ fundamentally replicates the same instructions for compilation that you would ordinarily use when compiling a program by hand. The benefit to a makefile, as you might have guessed, is that it speeds up the process by enabling many commands to be executed with a single command. For larger programs, the benefit is great as large numbers of files can be compiled together with only a single command. The example makefile below converts the code file, test.cpp, into an executable called test. It does this in two steps. The numbers in brackets are provided for clarity and do not form a part of the makefile.

```
[1] test: test.o
        g++ -o test test.o

[2] test.o: test.cpp
        g++ -c test.cpp
```

1. This segment is responsible for producing the executable file. The word before the colon, is the name of the executable you wish to create. What follows after the colon, are all of the object files you wish to use to construct the executable. You can combine many .o files in this way although one will typically have a main function.

2. This segment is responsible for converting a .cpp file into an object file. It uses the g++ -c flag to compile down the user code into object code.

Remember that your makefiles must include the make run instruction that allows them to run your programs. Makefiles without this instruction will result in no marks being awarded.

A final note on constructing makefiles is to be careful with the use of spaces. The compiler statements, those using g++ for example, are to be included on the line directly after those specify a target, must be a tab width from the left hand margin. If your makefile is not functioning, it is likely some of the spacing in the file is the issue.

3 Practical Activities

3.1 Activity 1: While Loops

In this activity, you are going to create your own code file, and makefile, in order to create a program that will create and execute a while loop running on some conditional value.

For this activity, you are required to create a makefile and a file called loop.cpp. All of the code required for the activity must be contained inside loop.cpp, including a main function that returns 0.

While loops are typically used when the condition that determines their end is not strictly tied to a known end point. Rather, they typically use sentinel variables that watch if the condition that stops the loop is true. This means that while loops are generally used in situations where the loop will end, but the exact point is not known beforehand. In practice, such loops are also constructed with a maximum limitation condition to prevent them from being infinite loops.

At the start of your program, you should prompt the user to enter in an integer. Once they have, your program should use a loop to determine the number of odd numbers from 0 to that number, inclusive of both values, and display it. The output should take the following format:

```
Enter an int: 43
Number of Odds: 23
```

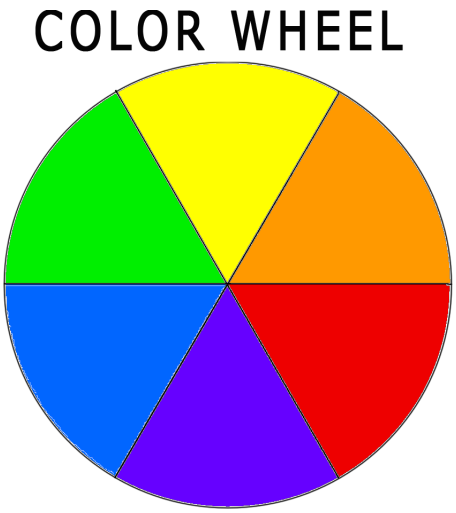
Remember to include a newline at the end of the final output.

You will have a maximum of 10 uploads for this activity.

3.2 Activity 2: Complex Operations

In this activity, you are going to create your own code file, colour.cpp, and makefile, in order to create a program that will respond with some more complex logic to user input.

Consider the following colour wheel:



The top three colours (from left to right) are green, yellow and orange. The bottom three (from left to right) are blue, purple and red.

Your program has to prompt the user to enter a colour from this list. If their input does not match any of the given colours, you are to print out an error message that reads as follows "Colour not found" with a new line at the end.

However if the colour is found then you need to prompt the user to input of an integer of a number between 1 and 3 (inclusive). You can presume the number will be within the correct range.

What your program does next depends on a number of conditions which are listed below:

- Input colour is a primary colour
  - Input Number is 1: Output (comma separated with no extra spaces) the list of all colours starting with yellow going clockwise.
  - Input Number is 2: Output the colour that was input.
  - Input Number is 3: Output the next primary colour (looking clockwise).
- Input colour is a secondary colour
  - Input Number is 1: Output (comma separated with no extra spaces) all the secondary colours starting with green going clockwise.
  - Input Number is 2: Output the input colour but converting the word to uppercase.
  - Input Number is 3: Output the next secondary colour (looking clockwise).

Unless otherwise stated, your outputs should always end with a newline.

Here are some examples of output. Follow these guidelines to format your own messages.

```
Input a colour: blellow
Colour not found
```

```
Input a colour: blue
Input a number: 3
yellow

Input a colour: green
Input a number: 1
green,orange,purple
```

If you require more information, you should visit the following link: https://en.wikipedia.org/wiki/Color_theory.

You are advised to use the cstring, sstream and string libraries for this task. You will have a maximum of 10 uploads for this activity.