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 1 Specifications

Release Date: 17-02-2025 at 06:00

Due Date: 21-02-2025 at 23:59

Late Deadline: 23-02-2025 at 23:59

Total Marks: 15

Contents

Task	Mark
Task 1	3
Task 2	3
Task 3	3
Task 4	3
Testing	3

Table 1: Mark Allocation

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually, no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as **no extension 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 classes).
- 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.
- Please note that there is a late deadline which is 48 hours after the initial deadline, but you will lose 20% of your mark that you achieved for the practical.

2 Overview

For this practical assignment, you will be putting your algorithmic thinking skills to the test, by aiding the Star-System Kingdom with their robot name encryption algorithm, **R**obot **O**bfuscation **B**amboozlement **O**scillation **T**ranslator (ROBOT).

The ROBOT algorithm is a 4 stage algorithm and can be expressed by Figure 1 and Algorithm 1, which has been implemented for you.

Your task will be to develop four sub-algorithms which represent each of the stages in Algorithm 1. The exact details of the sub-algorithms are discussed in Section 3.1. Using the algorithms you developed for each of the subsections, ask any AI Chat Bot of your choosing to generate the C++ code for the algorithm. Note, the purpose of this practical is to practice algorithmic thinking which is critical to the passing of COS132, and all future COS modules. Please ensure that you are able to successfully develop abstract algorithms for your semester tests and exams.

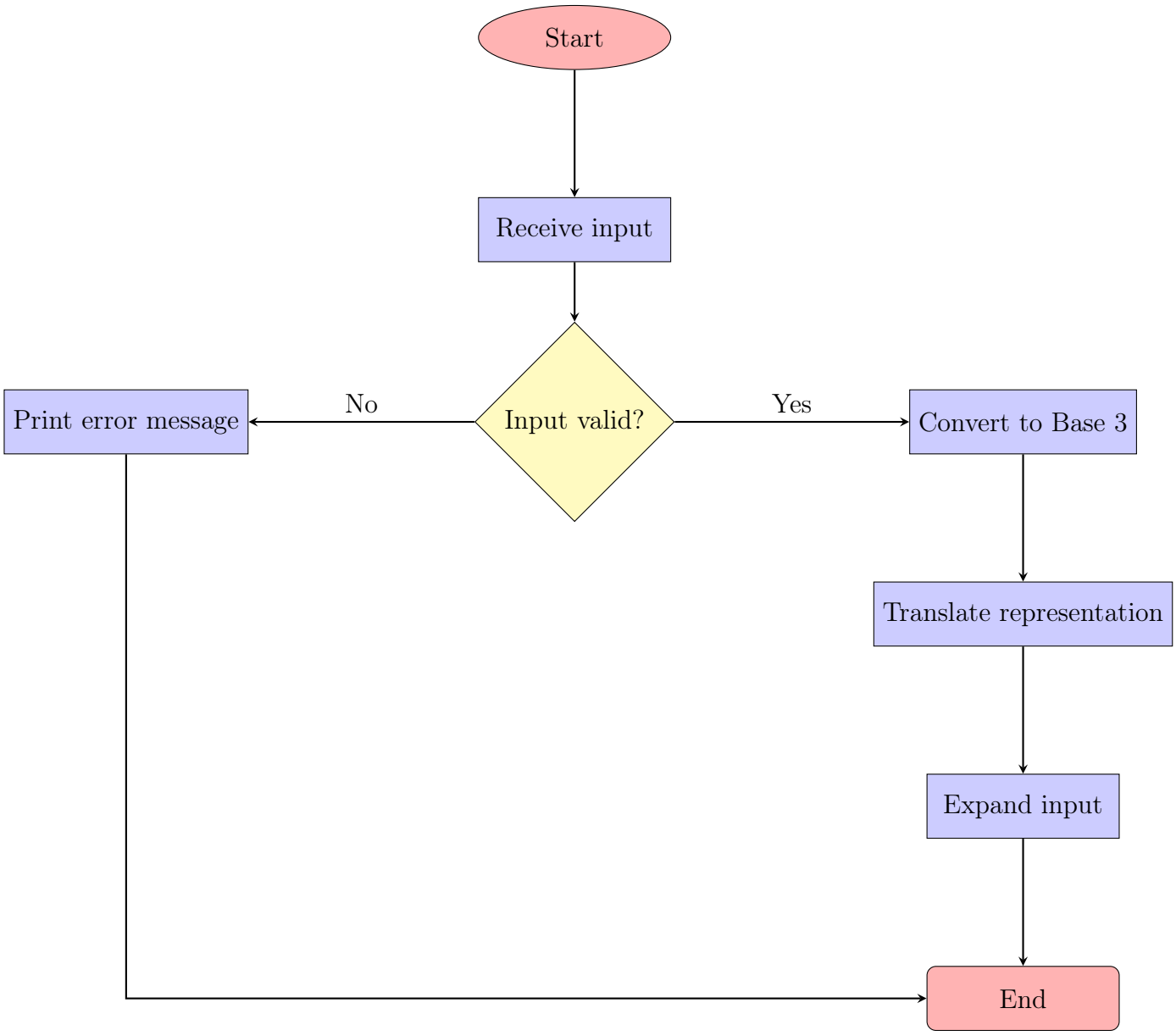

Figure 1: ROBOT Algorithm

Algorithm 1 ROBOT Algorithm

Require: `input` from user/external source
  if `input` is valid **then**
    `convertedInput` = convert `input` into Base 3 representation.
    `translatedInput` = translate the `convertedInput` into an alternative representation.
    `ExpandedInput` = expand the `translatedInput`.
  else
    print out a fitting error message.
  end if

Table 2 contains a mapping of stage number to task number which is important for the following section. Note due to the required use of AI for this practical, adaptive marking will be used. You will also be allowed to see your output such that you can determine where your algorithm failed. This will not be the case for the remaining practicals.

3 Your Task:

You are required to implement all of the functions laid out in this section. Each stage in Algorithm 1 should be completed in the corresponding task.

Stage Nr	Task Nr	Stage Line
1	1	if input is valid
2	2	convert `input` into Base 3 representation
3	3	translate the `convertedInput` into an alternative representation.
4	4	expand the `translatedInput`.

Table 2: Stage to Task Mapping

3.1 ROBOT

A namespace is a way to group variables, functions, classes and more into a single collective "entity" which can be used to refer to the "entity". Later in this course, you will learn more about different namespaces. For this assignment a namespace, called ROBOT, has been defined for you. This namespace contains 4 functions named `task1, task2, task3,` and `task4`. The AI Generated code for the algorithms you will develop should be copied into the relative task. A series of examples are provided for you in Section 4.

3.1.1 Functions

This section discusses the tasks. Each task should be completed as a function. Informally, a function is a collection of statements which can take input and return a result. You will learn about the wonders of functions during the semester.

- `bool task1(std::string input)`
  - This function needs to validate the `input`.
  - Input is considered to be valid if it only consists of the characters A to Z and 0 to 9.
  - Special characters or lowercase characters are considered invalid.
  - If the `input` is valid the function should return true, else it should return false.
- `std::string task2(std::string input)`
  - This function needs to convert each character in the input to a Base 3 representation, and return this Base 3 representation.
  - Use Table 3 to perform the conversion.

- `std::string task3(std::string convertedInput)`
  - This function needs to translate the converted input.
  - This is done using the mapping in Table 4. Each character in the `convertedInput` needs to be translated using this table to form the result.
- `std::string task4(std::string translatedInput)`
  - This function needs to expand the translated input.
  - This is done by counting the number of consecutive symbols next to each other and replacing those symbols by a code.
  - This code is formulated by placing the count and a symbol next to each other.
  - Example: AAAA becomes 4A and ACCB becomes 1A2C1B.

4 Examples

These examples were obtained using the provided `ROBOT` algorithm's output.

4.1 Valid Example 1

```
Enter droid name: L3
Input was valid... progressing to next phase
Converted: 01021002
Translated: ABACBAAC
Expanded: 1A1B1A1C1B2A1C
```

4.2 Valid Example 2

```
Enter droid name: G5N91
Input was valid... progressing to next phase
Converted: 00201011011110221000
Translated: AACABABBABBBBACCBAAA
Expanded: 2A1C1A1B1A2B1A4B1A2C1B3A
```

4.3 Valid Example 3

```
Enter droid name: 9JD05D7O2
Input was valid... progressing to next phase
Converted: 10220100001002221011001010200112100 1
Translated: BACCABAAAABAACCCBABBAABABACAABBCBAAB
Expanded: 1B1A2C1A1B4A1B2A3C1B1A2B2A1B1A1B1A1C2A2B1C1B2A1B
```

Character	Base 3 Codes
A	0000
B	0001
C	0002
D	0010
E	0011
F	0012
G	0020
H	0021
I	0022
J	0100
K	0101
L	0102
M	0110
N	0111
O	0112
P	0120
Q	0121
R	0122
S	0200
T	0201
U	0202
V	0210
W	0211
X	0212
Y	0220
Z	0221
0	0222
1	1000
2	1001
3	1002
4	1010
5	1011
6	1012
7	1020
8	1021
9	1022

Table 3: Trinary Code Representation of Characters

Numerical Value	Symbol
0	A
1	B
2	C

Table 4: Translation Mapping

4.4 Invalid Example 1

```
Enter droid name: Yr
Input was invalid... halting execution
```

4.5 Invalid Example 2

```
Enter droid name: JlKr(3b
Input was invalid... halting execution
```

4.6 Invalid Example 3

```
Enter droid name: q8*)KG7
Input was invalid... halting execution
```

Ensure that when you run your code locally, you get the same results. There should be no extra debugging printouts. Ensure that the AI chatbot did not add any additional printouts.

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 [1] 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 code size.

The mark you will receive for the testing coverage task is determined using Table 5:

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

Coverage ratio range	Testing mark
0%-30%	0
30%-60%	1
60%-90%	2
90%-100%	3

Table 5: 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.

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:
  - string
  - sstream
  - iostream
- You are supplied with a **trivial** main demonstrating the basic functionality of this assessment.

7 Upload Checklist

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

- `ROBOTS.cpp`
- `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 *.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 mark will be the mark you obtain for this practical. Upload your archive to the Practical 1 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**