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

COS110 - Program Design: Introduction

Practical 5 Specifications

Release Date: 01-09-2025 at 06:00

Due Date: 05-09-2025 at 23:59

Late Deadline: 06-09-2025 at 00:59

Total Marks: 109

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 1 hour 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

Esoteric programming languages are programming languages designed as a proof of concept to test the boundaries of programming. This most often leads to languages designed as a joke, but sometimes gain popularity due to how obscure the language is. An example of this is Brainf*ck. This is a language with only 8 valid symbols that can be used in the program and uses a tape to store all memory. You will use c++ operators to write an interpreter for this language.

3 Your Task:

You are required to implement the following class diagram illustrated in Figure 1. Pay close attention to the function signatures as the `h` files will be overwritten, thus failure to comply with the UML, will result in a mark of 0.

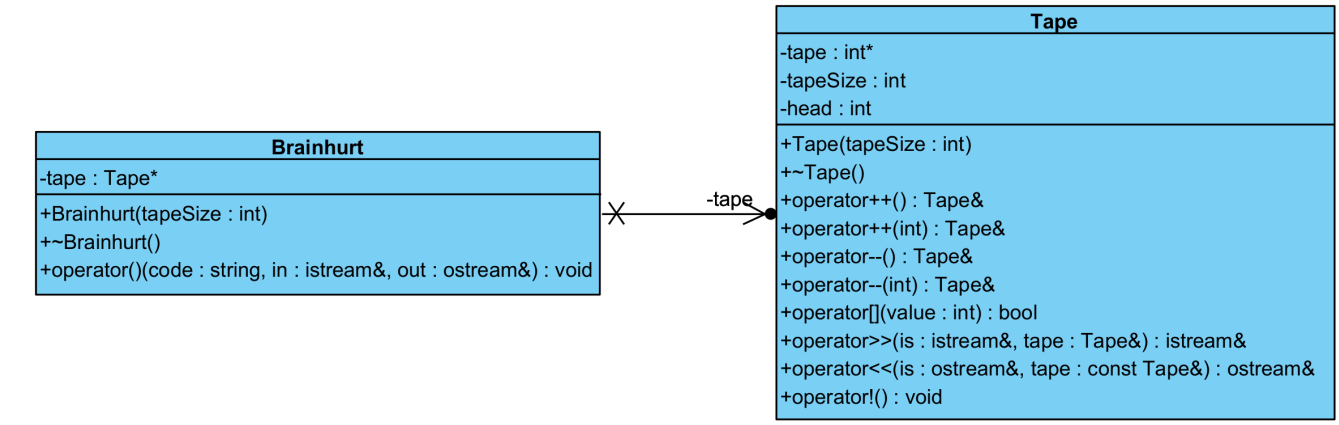

Figure 1: Class diagrams

Note, the importance of the arrows between the classes are not important for COS 110 and will be expanded on in COS 214.

3.1 Tape

This class will store and manage an array of integers representing a tape that will be used to write a brainf*ck interpreter.

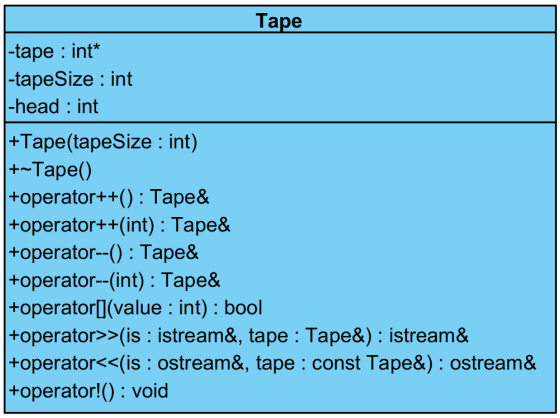

Figure 2: Tape UML

3.1.1 Members

- Member 1
  - This is the values stored in the tape.
- Member 2
  - This is the size of the tape, and also the size of the array.
- Member 3
  - This can be thought of as a pointer showing the current active cell on the tape.

3.1.2 Functions

- Function 1
  - Initialise the tape array using the passed-in size.
  - All values on the tape should be initailised to 0.
  - The head should point the first cell on the tape (at index 0).
- Function 2
  - Deallocate all dynamic memory of the class.
- Function 3
  - Increment the value at the head of the tape.
  - Return a reference to the current object.

- Function 4
  - If the head is at the last cell in the tape then throw the exception "end of tape reached".
  - Otherwise move the head pointer one cell forward.
  - Return a reference to the current object.
- Function 5
  - Decrement the value at the head of the tape.
  - Return a reference to the current object.
- Function 6
  - If the head is at the first cell in the tape then throw the exception "front of tape reached".
  - Otherwise move the head pointer one cell backwards.
  - Return a reference to the current object.
- Function 7
  - This is a constant function.
  - Return true if the value at the head is equal to the passed-in value. Return false otherwise.
- Function 8
  - This is a friend function.
  - Try to pipe an integer from the input stream. *Hint: Use the .good() function on the inputstream to check whether the piping was successful.* If an invalid integer was piped then throw the exception "Invalid input".
  - If a valid integer was piped, then store this value in the cell pointed to by head.
- Function 9
  - This is a friend function.
  - This will output the value, in the cell pointed to by the head, to the ostream object.
  - The format is as follows:
    - Convert the integer to a character and output this character.
    - After the character output a space chracater.
    - Then output the integer value inside brackets.
    - Lastly output an endline character.
- Function 10
  - Reset the tape, by setting all cells to 0.

3.2 Brainhurt

This class will implement an interpreter for the Brainf*ck language. This interpreter does not allow negative cell values or comments in code, so be careful when copying example programmes from the Internet.

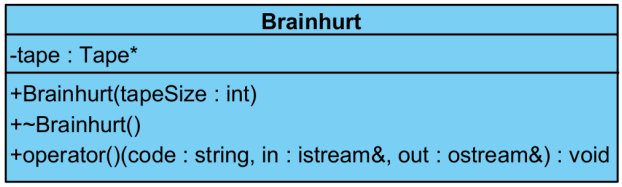

Figure 3: Brainhurt UML

3.2.1 Members

- Member 1
  - This is the tape used by the interpreter.

3.2.2 Functions

- Function 1
  - Initialise the tape object using the passed-in size.
- Function 2
  - Deallocate all dynamic memory of the class.
- Function 3
  - This function will take in Brainf*ck code, run it and give the output of the program.
  - Start by resetting the tape such that all cells contain 0.
  - Loop through the passed-in code. Every character is a instruction that you should then run. The instructions are explained on the next page.
  - The process of looping through the code should by done in a try-catch block. Catch all `const char*` exceptions and process them by outputting the following to the ostream object:
    - `Interpreter failed on instruction % with error: %`
    - The first % should be replaced with the instruction in the code that caused the error.
    - The second % should be replaced with the value in the exception.
  - After the message is outputted, return out of the function to stop executing the code.

* Brainf*ck rules:

1. `<`
   * Move the head of the tape one space to the left.
2. `>`
   * Move the head of the tape one space to the right.
3. `+`
   * Increment the value at the head of the tape.
4. `-`
   * Decrement the value at the head of the tape.
5. `.`
   * Output the value at the head of the tape to the ostream object.
6. `,`
   * Input the value for the head of the tape from the istream object.
7. `[`
   * If the value at the head cell is 0, then move forward to the instruction after the matching `]`.
   * If the value at the head cell is not 0, then go to the next instruction.
   * Example:
     - Given the program `,>,[<+>-]<.`
     - If you are at the `[` command, and the value at the head pointer is 0, then you have to move to the instruction after the `]` command. Thus it will be at the part of the code that says `<.`.
     - Note that nested loops are possible. You have to accomodate for this by skipping over pairs of `[]` such that you find the correct closing bracket.
     - If you can't find the matching closing bracket, loop to the end of the code, and then throw the exception "Invalid loop".
8. `]`
   * If the value at the head cell is 0, then go to the next instruction.
   * If the value at the head cell is not 0, then move backward to the instruction after the matching `[`.
   * Example:
     - Given the program `,>,[<+>-]<.`
     - If you are at the `]` command, and the value at the head pointer is not 0, then you have to move to the instruction after the `[` command. Thus it will be at the part of the code that says `<+>-`.
     - If you can't find the matching closing bracket, loop to the start of the code, and then throw the exception "Invalid loop".
9. Any other character
   * If you encounter any other character in the code, then throw the exception "Illegal character in code".

4 Memory Management

As memory management is a core part of COS110 and C++, each task on FitchFork will allocate approximately 10% of the marks to memory management. The following command is used:

```
valgrind --leak-check=full ./main
```

Please ensure, at all times, that your code *correctly* de-allocates *all* the memory that was allocated.

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 class size.

The mark you will receive for the testing coverage task is determined using Table 1:

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 1: 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

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

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.
- You may only use the following libraries:
  - iostream
  - sstream
  - string
- 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:

- Tape.cpp
- Brainhurt.cpp
- `main.cpp`
- Any textfiles used by your `main.cpp`
- `testingFramework.h` and `testingFramework.cpp` if you used these files.

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++ -Werror -Wall *.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 10 submissions and your best mark will be your final mark. Upload your archive to the Practical 5 slot on the FitchFork website. If you submit after the deadline but before the late deadline, a 20% mark deduction will be applied. **No submissions after the late deadline will be accepted!**