COMP 2012 Object-Oriented Programming and Data Structures

Assignment 2 LegoGPT

Image Source: https://www.designrush.com/

Introduction

Suppose you are want to build your own chat robot like the **ChatGPT**. However, training models starting from scratch takes years of time and huge amount of resources. Let's try to build your own version by combining the resources you have.

Suppose you have several small modules that can answer different topics or can translate languages. You can combine them together to have a stronger chatbot, like you are playing with Lego. For example, you have a simple module that can answer **math** questions in English and another module that can translate **foreign language** into English. By combining these two modules together, you can translate **math** questions in **foreign language** into English, and then pass the "**math** questions in English" to your **math** module. In the end, you have a new module that can answer **math** question in any **foreign language**.

LegoGPT

In this assignment, we are going to build LegoGPT, which is a container of three different modules:

- **Handler Module** handles questions in a certain topic. For example, math question asked in the default language (English). If one module cannot answer the question, it simply replies "Sorry, I don't know".
- **Filter Module** filters the input language into target language. In this example, we only consider two kinds of filters: language and tone. A language filter is similar to a translator for the handler module, and bound to them. When a *question* comes in to a Handler Module, it first passes the *question* to the language filter it is bound to. The language filter translates the *question* to English if it can; otherwise, it does nothing to the *question*. Then the Handler Module handles the *question*, and generates an *answer* (the *answer* can be "Sorry, I don't know"). Finally, the Handler Module passes the *answer* to its language filter again, and the language filter translates the *answer* to the target language if possible. Similarly, the tone filter translate within the same language, but with different tones. For example, a **humor** tone filter can change official answers to humorous answers.
- **FilterHandler Module** works is the combination of a **Handler Module** and a **Filter Module**. In other words, it can both handle questions in a certain topic, and can filter *questions* and *answers* for other **Handler Modules**. For example, humor is one **FilterHandler Module**. It can directly answer questions like "Tell me a joke", it can also filter other modules' answers to humorous answers. The class of `FilterHandlerModule` is defined by ***multiple inheritance*** of class `FilterModule` and class `HandlerModule`. Since ***multiple inheritance*** will not be covered by this course, everything for `FilterHandlerModule` is provided for you. As long as you implemented class `FilterModule` and class `HandlerModule` correctly, `FilterHandlerModule` should also work correctly. If you are interested in ***multiple inheritance***, you can learn about it by yourself in this link.

Data structures

In this assignment, we use two data structures: dynamic **array** and dynamic **map**. We have introduced and practiced dynamic **array** before, and we only introduce dynamic map here. **Map** stores key-value pairs without duplicated keys. **Map** supports the "find" function: "find" inputs the key and return the corresponding value in the data structure. In this assignment, we will practice a very basic **Map** by implementing with two arrays of the same order: the first array stores the keys and the second array stores the corresponding values. For the "find" function, we first traverse all values in the array of keys, if we find the key, return the value in

the same index; otherwise, we return a `nullptr` or an empty `string` based on the type of the value.

Classes overview

The following section describes the given classes.

DataMap

```
class DataMap{
  public:
    DataMap();
    ~DataMap();
    int size() const { return arraySize; }
    void insert(const string &key, const string &value);
    string find(const string &key) const;
    void clear();

  private:
    int arraySize;
    string *keys;
    string *values;
};
```

Instances of DataMap class are members of Handler Module, Filter Module and FilterHandler Module to store data. You need to implement this class first so that you can move on to other parts. A similar class `Array` is implemented and provided to you for your reference.

Note that you should implement it in a *dynamic* way: the size should be exactly the same as the number of key/value pairs stored. For example, if you store 2 key/value pairs, the `size()` should return 2.

Module

```
class Module{
  public:
    Module(const string &name) : m_name(name){};
    virtual ~Module(){};
    virtual void print() const = 0;
    void setName(const string &newName) { m_name = newName; }
    string getName() { return m_name; }

  protected:
    string m_name;
};
```

The `Module` class is the base class of both `FilterModule` and `HandlerModule`. Note that `print()` is set as pure virtual function, which means the derived classes must implement the function `print()` themselves.

HandlerModule

```
class HandlerModule : public Module{
  public:
    HandlerModule(const string &name, const string &topic) : Module(name), topic(topic)
    {
      this->languageFilter = nullptr;
      this->toneFilter = nullptr;
    };
    //-------You need to define destructor and print for HandlerModule-------
    //-------You are only supposed to modify code below-------

    // helper information: std::cout << "Module name = " << m_name << ", this is a Handler, can handle topic = "
    //-------You are only supposed to modify code above-------
    string getTopic() const { return topic; }
    void trainData(const string &question, const string &answer);
    string answer(const string &question) const;
    void setLanguageFilter(FilterModule *input) { this->languageFilter = input; }
    void setToneFilter(FilterModule *input) { this->toneFilter = input; }

  protected:
    string inneranswer(const string &question) const;

    string topic;
    FilterModule *languageFilter;
    FilterModule *toneFilter;
    DataMap data;
};
```

You need to implement the destructor function and `print()` function. The `HandlerModule` class stores the trained data in `data` which is a map between *question* and *answer*. Examples are shown in Test case 2.

`HandlerModule` has two `FilterModule` pointer members, one for a language filter and the other one for tone filter. Given a *question*, the `HandlerModule` instance does the following steps:

- Firstly, it passes the *question* to the language filter if `languageFilter` is not a `nullptr`.
- Secondly, it passes the "translated" *question* to the tone filter if `toneFilter` is not a `nullptr`.
- Thirdly, it queries the `inneranswer` function, which find the *answer* within its own `data`. If it cannot find any answer, it simply replies `"Sorry, I don't know"`.
- Fourthly, it passes the *answer* to the tone filter if `toneFilter` is not a `nullptr`.
- Lastly, it passes the *answer* to the language filter if `languageFilter` is not a `nullptr`.
  Note that even if `toneFilter` is not a `nullptr`, it may still does nothing to your *question* or *answer*. For example, a "humor" filter will only change the tone for your answer, not the question. In the end, we aim to "ask a math joke in a foreign language, and get the reply also in foreign language".

You are *not allowed* to change provided definitions. You are *not allowed* to call `inneranswer()` outside this `HandlerModule` class. `inneranswer()` is only supposed to be called in `answer()`.

You can use the provided helper information to have the exact same format for `print()` function.

FilterModule

```
class FilterModule : public Module{
    public:
      FilterModule(const string &name, const FilterType type) : Module(name), type(type){};
      //-------You need to define destructor and print for HandlerModule-------
      //-------You are only supposed to modify code below-------

      // helper information: std::cout << "Module name = " << m_name << ", this is a filter, filter type = " << g
      //-------You are only supposed to modify code above-------

      FilterType getType() const { return type; }
      void trainPreData(const string &source, const string &target);
      void trainPostData(const string &source, const string &target);
      string translatePreData(string question) const;
      string translatePostData(string question) const;

    protected:
      FilterType type;
      DataMap preData;
      DataMap postData;
};
```

You need to add definitions for the destructor function and `print()` function in the header file. You also need to implement all the functions defined without adding any helpers functions. The `FilterModule` class stores the trained data in `preData` and `postData` which are maps between *question* and *answer*. `preData` stores the data for translation of handlers's questions and `postData` stores the data for translation of handlers's answers. In other words, `preData` is used in the first two steps of handlers's `answer` and `postData` is used in the last two steps of handlers's `answer`.

Recall that if the `preData` or `postData` does not have the corresponding information, the filter does not change the *question* or the *answer*. For example, if one answer is "3 + 5 = 8" and `postData` contains nothing for this string, then the final answer is still "3 + 5 = 8".

You can use the provided helper information to have the exact same format for `print()` function.

LegoGPT

```
class LegoGPT{
  public:
    LegoGPT(){};
    ~LegoGPT();
    void LoadHandlerModule(HandlerModule &input);
    void LoadFilterModule(FilterModule &input);
    void printHandlers() const;
    void printFilters() const;
    void printTopics() const;
    void printLanguages() const;
    void clear();
    void chat(string topic, string question) const;

  private:
    HandlerMap handlerMap;
    FilterArray filterArray;
    Array topics;
    Array languages;
};
```

LegoGPT stores all the handlers' pointers in the `HandlerMap` and stores all the filters' pointers in the `FilterArray`. Besides, it has two arrays: one for all supported topics and the other one for all supported languages.

When loading a handler module, the LegoGPT stores the pointer of the handler module into `handlerMap` and stores the topic of the module to `topics`. For simplicity, we will never store the same handler module more than once in test cases, so you don't

need to consider loading same topics more than once for the `topics`.

When loading a filter module, the LegoGPT stores the pointer of the filter module into `filterArray` and stores the language if the filter is a language filter to `languages`. For simplicity, we will never store the same filter module more than once in test cases, so you don't need to consider loading same language more than once for the `languages`.

```
void LegoGPT::chat(string topic, string question) const {
  if (nullptr == this->handlerMap.find(topic)) {
    cout << "Sorry, topic not supported. Please look for supported topic." << endl;
    return;
  }
  cout << this->handlerMap.find(topic)->answer(question) << endl;
  return;
}
```

We have implemented the `chat` function for you. It first searches if the `topic` exists in the `handlerMap`. If not exists, it simply

We value academic integrity very highly. Please read the Honor Code section on our course webpage to ensure you understand what is considered plagiarism and the penalties. The following are some of the highlights:

- Do NOT try your "luck" - we use sophisticated plagiarism detection software to find cheaters. We also review codes for potential cases manually.
- The penalty (for **BOTH** the copier and the copiee) is not just getting a zero in your assignment. Please read the Honor Code thoroughly.
- Serious offenders will fail the course immediately, and there may be additional disciplinary actions from the department and university, including expulsion.

End of Introduction

and again without any errors.

Task description

We summarize all tasks as following:

- Todo 1: Implement "DataMap" in "array.cpp".
- Todo 2: Add function declaration in "handler.h" and implement them in "handler.cpp": (destructor function and print() function)
- Todo 3: Implement "trainData" and "answer" function of "HandlerModule".
- Todo 4: Add function declaration in `filter.h` and implement them in `filter.cpp`: (destructor function and print() function)
- Todo 5: Implement "trainPreData", "trainPostData", "translatePreData", "translatePostData" in "filter.cpp".
- Todo 6: Implement "LoadHandlerModule", "LoadFilterModule" in "legoGPT.cpp. Implement "HandlerMap", "FilterArray" in "handler.cpp", "filter.cpp".
- Todo 7: Implement `~LegoGPT()` and `void LegoGPT:: clear()`

Generally, you will need to complete the tasks in 3 new files: **array.cpp**, **filter.cpp** and **handler.cpp**. You need to complete the **filter.h** and **handler.h** by adding the definitions of destructor and `print()` correctly. You also need to finish all missing functions in **legoGPT.cpp** according to definitions. In addition, **you are not allowed to include additional libraries unless specified**. You may only `#include` any given header files (**array.h**, **filter.h**, **handler.h**, **filterhandler.h**, **legoGPT.h** and **module.h**). You are also allowed to include **<iostream>** for debug purposes.

We provide many test cases in the `main.cpp` and the following are their descriptions. Note that we have some additional hidden test cases to make sure your code is not "hard-coded".

Test case 1

Test `FilterModule`, `HandlerModule` and `FilterHandlerModule` and print the information.

Task case 2

Based on Test case 1, finish the implementation of LegoGPT and allows training data for handlers. The question is directly sent to

the handler.

Task case 3

Based on Test case 2, train a simple math handler, and load it into LegoGPT. The question is directly sent to the LegoGPT.

Task case 4

Based on Test case 3, train a simple math handler, and load it into LegoGPT. But ask a question with topic "physics".

Task case 5

Based on Test case 4, train a simple math handler, a Martian filter (translate between English and language on Mars:) and a humor FilterHandler. We need to slice the humor to both `FilterModule` and `HandlerModule`. Load them into LegoGPT. Then print topics and languages, and finally ask a math question and ask for a joke.

Task case 6

Based on Test case 5, bound the humor filter with math handler and get a humorous math answer. Then bound the Martian filter with math handler and ask a math question in Martian.

Task case 7

Based on Test case 6, bound the Martian filter with humor handler and ask for a joke in Martian.

Task case 8

Based on Test case 7, Let's ask for a math joke in Martian!

End of Description

Resources & Sample Output

- Skeleton code: skeleton.zip
- Demo program (executable for Windows): demo.exe
- Demo program (executable for Mac): demomac.exe

```
Welcome to PA2, basic tests for Module
Module name = basic filter, this is a filter, filter type = reserved
Module name = basic handler, this is a Handler, can handle topic = naiveHandler
Module name = basic filterHandler, this is a filterHandler, filter type = reserved, handler topic =
naiveFilterHandler
-------------------------end of test 1------------------------
-------------------------begin of test 2------------------------
There is(are) 1 handler(s) in the LegoGPT
They are:
naiveHandler
There is(are) 1 filter(s) in the LegoGPT
Module name = basic filter, this is a filter, filter type = reserved
There is(are) 2 handler(s) in the LegoGPT
They are:
naiveHandler
naiveFilterHandler
There is(are) 2 filter(s) in the LegoGPT
Module name = basic filter, this is a filter, filter type = reserved
Module name = basic filterHandler, this is a filterHandler, filter type = reserved, handler topic =
naiveFilterHandler
Let's train simple handlers!
Question is: How are you? answer: Fine, thank you, and you?
-------------------------end of test 2------------------------
-------------------------begin of test 3------------------------
Basic tests for LegoGPT
Question is What is 3 + 5
3 + 5 = 8
-------------------------end of test 3------------------------
-------------------------begin of test 4------------------------
Question is What is 3 + 5
3 + 5 = 8
Topic is physics
Sorry, topic not supported. Please look for supported topic.
-------------------------end of test 4------------------------
-------------------------begin of test 5------------------------
There is(are) 2 topic(s) in the LegoGPT
They are:
math
humor

There is(are) 1 additional language(s) in the LegoGPT
They are:
Martian filter

3 + 5 = 8
- Two strings walk into a bar. The bartender says, "So what'll it be?" The first string says, "I think
I'll have a beer^CjfdLk jk3s d#f67howe%^U r89nvy~~owmc63^Dz x.xvcu" "Please excuse my friend," the
second string says, "He isn't null-terminated."
-------------------------end of test 5------------------------
-------------------------begin of test 6------------------------
Let's get a humorous math answer.
I don't know what is 3 + 5, but I know 3 + 5 = 5 + 3. Because Integers under Addition form an Abelian
Group.
Let's ask a math question in Martian.
Before training:
Sorry, I don't know
```

End of Download

```
Group.
                        end of test 6
```

Testing

Since we no longer provide public test cases on ZINC this semester, you can perform testing on your local machine by compiling with the given Makefile. If all tasks have been implemented correctly, the generated program **main.exe** should have the same

output as the sample above.

End of Testing

```
second string says, "He isn't hurt terminated."
-------------------------end of test 7-----------------------
```

Submission and Deadline

Deadline: 23:59:00 on *April 12, 2023*.

ZINC Submission

Please submit the following files to ZINC by zipping the following 6 files. ZINC usage instructions can be found here.

```
array.cpp
filter.cpp
filter.h
handler.cpp
handler.h
legoGPT.cpp
```

Notes:

- The compiler used on ZINC is `g++11`. However, there shouldn't be any noticeable differences if you use other versions of the compiler to test on your local machine.
- **We will check for memory leak in the final grading**, so make sure your code does not have any memory leak.
- You may submit your file multiple times, but only the latest version will be graded.
- Submit early to avoid any last-minute problems. Only ZINC submissions will be accepted.
- If you have encountered any server-side problems or webpage glitches with ZINC, you may post on the ZINC support forum to get attention and help from the ZINC team quickly and directly. If you post on Piazza, you may not get the fastest response as we need to forward your report to them, and then forward their reply to you, etc.

Compilation Requirement

It is **required** that your submissions can be compiled and run successfully in our online autograder ZINC. If we cannot even compile your work, it won't be graded. Therefore, for parts you cannot finish, just put in a dummy implementation so that your whole program can be compiled for ZINC to grade the other parts you have done. Empty implementations can be like:

```
int SomeClass::SomeFunctionICannotFinishRightNow()
{
  return 0;
}

void SomeClass::SomeFunctionICannotFinishRightNowButIWantOtherPartsGraded()
{
}
```

Reminders

Make sure you actually upload the correct version of your source files - we only grade what you upload. Some students in the past submitted an empty file or a wrong file or an exe file which is worth zero mark. So **you must double-check the file you have submitted**.

Late Submission Policy

There will be a penalty of -1 point (out of a maximum 100 points) for every minute you are late. For instance, since the deadline for assignment 2 is 23:59:00 on *April 12*, if you submit your solution at 1:00:00 on *April 13*, there will be a penalty of -61 points for your assignment. However, the lowest grade you may get from an assignment is zero: any negative score after the deduction due to late penalty (and any other penalties) will be reset to zero.

End of Submission and Deadline

Frequently Asked Questions

- 1. `printHandlers` prints handlers' topics. The “names” in HandlerMap stores the topic of handlers.

End of Frequently Asked Questions

Menu

Page maintained by

Jiajun Xin
Email: jxin@cse.ust.hk

Last Modified: 03/28/2023 14:23:02

Homepage

Course Homepage