2024 年ソフトウェア演習2B

第 1 回課題

B243392　ALCANDER IMAWAN　2024 年 6 月 12 日

Q1

プログラム

Message.h

```
#include <iostream>

class Message {

private:
        char* message;

public:
        Message(); // Constructer
          ~Message(); //Destructor

          void setMessage(const char* message);
          char* getMessage(void);
};
```

Message.cpp

```
#include "Message.h"
#include <cstring> //for std::strlen and std::strcpy

// Constructer initializing
Message::Message(){
        message = nullptr;
}
```

```

//Destructor
Message::~Message(){
        if(message != nullptr)
        delete[] message;
}

//this is the function to set Message,
//it works by making a new char* with (msg + 1) as its length
//then copy msg to message with strcpy
void Message::setMessage(const char* msg){
        message = new char[std::strlen(msg) + 1];
        std::strcpy(message, msg);
}

//return message
char* Message::getMessage(void){
        return message;
}

```

main.cpp

```
#include "Message.h"

int main (int argc, char *argv[]){
        //make a new Message object called obj
        Message obj;
        obj.setMessage("Hello World.");
        std::cout << obj.getMessage() << std::endl;

        return 0;
}
```

動作確認

```
[a243392@xdev07 q1]$ ./q1
Hello World.
[a243392@xdev07 q1]$
```

Q2

プログラム

Message.h

```
#include <iostream>

class Message {

private:
    char* message;

public:
    Message(); // Constructer
    ~Message(); //Destructor

    void setMessage(const char* message);
    const char* getMessage(void) const;

    //declaration of stream operators
    friend std::istream& operator>>(std::istream& stream, Message& obj);
    friend std::ostream& operator<<(std::ostream& stream, const Message& obj);
};


```

Message.cpp

```
#include "Message.h"
#include <cstring> //for std::strlen and std::strcpy

// Constructer initializing
Message::Message(){
    message = nullptr;
}

//definition of extraction operator (>>)
std::istream& operator>>(std::istream& stream, Message& obj){
	//temporary buffer to hold the input, by using buffer, we have better memory management,
	//safer and more robust(prevent overflows), and dynamic memory allocation
	char buffer[1024];
	stream.getline(buffer, 1024);

	//set the message using setMessage function below
	obj.setMessage(buffer);

	return stream;
}

//definition of the insertion operator(<<)
std::ostream& operator<<(std::ostream& stream, const Message& obj){
	if(obj.getMessage() != nullptr){
		stream << obj.getMessage();
	}
	return stream;
}

//Destructor
Message::~Message(){
	if(message != nullptr)
	delete[] message;
}

//this is the function to set Message,
//it works by making a new char* with (msg + 1) as its length
//then copy msg to message with strcpy
```

```
void Message::setMessage(const char* msg){
    message = new char[std::strlen(msg) + 1];
    std::strcpy(message, msg);
}

//return message
const char* Message::getMessage(void) const {
    return message;
}

```

main.cpp

```
#include "Message.h"

int main (int argc, char *argv[]){
    //make a new Message object called obj
    Message obj;
    std::cout << "Input message: ";
    //use >> operator to input to Message object
    std::cin >> obj;
    std::cout << "Output message:" << std::endl;
    //use << operator to output from Message object
    std::cout << obj << std::endl;

    return 0;
}
```

動作確認

```
[a243392@xdev07 q2]$ ./q2
Input message: this is a test for q2
Output message:
this is a test for q2
[a243392@xdev07 q2]$
```

Q3

プログラム

Message.h と Message.cpp は Q2 と同じプログラムを使っている。

RepeatMessage.h

```
#include <iostream>
#include "Message.h"

//class RepeatMessage is instanced from Message class,
//and all public members of Message class is accessible by RepeatMessage class
class RepeatMessage: public Message {

private:
        char* message;
        int nloops;

public:
        RepeatMessage(int nloops);
        ~RepeatMessage();
        const int getNloops()const;
        //overload (<<) operator for RepeatMessage class
        friend std::ostream &operator<<(std::ostream& stream, const RepeatMessage& obj);
};

```

RepeatMessage.cpp

```
#include "RepeatMessage.h"
#include <cstring> //for std::strlen and std::strcpy

//constructor implementation
RepeatMessage::RepeatMessage(int n): Message(), nloops(n){} // Constructer with nloops

```

```
//definition of the insertion operator(<<) for RepeatMessage
std::ostream &operator<<(std::ostream &stream, const RepeatMessage &obj){
    if(obj.getMessage() != nullptr){
        for(int i = 0; i < obj.getNloops(); i++){
            stream << obj.getMessage();
        }
        stream << std::endl;
    }
    return stream;
}

//function to get the nloops
const int RepeatMessage::getNloops()const{
    return nloops;
}

//Destructor
RepeatMessage::~RepeatMessage(){
    if(message != nullptr)
    delete[] message;
}
```

main.cpp

```
#include "RepeatMessage.h"

int main (int argc, char *argv[]){
    //make a new Message object called obj
    RepeatMessage obj(3);
    std::cout << "Input message: ";
    std::cin >> obj;
    std::cout << "Output message:" << std::endl;
    std::cout << obj;

```

```
        return 0;
}
```

動作確認

```
[a243392@xdev07 q3]$ ./q3
Input message: This is a message␣␣
Output message:
This is a message␣␣This is a message␣␣This is a message␣␣
[a243392@xdev07 q3]$
```

自己チェック項目

以下の項目について, 1 から 4 までの 4 段階で自己評価しなさい.

4. 十分に理解した 3. 少し不安が残るが理解した 2. 十分には理解できていない 1. まったく理解できない

4 クラスの実装の仕方を理解した.

4 private, protected, public などのアクセス指定子の意味と使い方を理解した.

4 コンストラクタ, デストラクタの実装方法を理解した.

4 メンバ変数の実装方法を理解した.

4 クラスに対する演算子の実装方法を理解した.

3 変数の参照渡しについて理解し, 値渡しとの違いを説明できる.

4 クラスの継承について理解し, 既存クラスを継承した別のクラスを実装することができる.

4 インデント(字下げ)など, 一貫したスタイルでプログラムが書ける.

4 プログラムに適切なコメントを入れることができる.

4 適切な変数名を用いることができる.