ソフトウェア演習２B

クラスの作成と継承

クラス

- C言語の構造体を発展させたようなもの
  - メンバ変数とメンバ関数を持つ
  - 構造体とは違い、クラスのインスタンスが生成されたときに自動的に呼び出されるコンストラクタとインスタンスが使われなくなったときに呼び出されるデストラクタと呼ばれる関数を定義できる。
- クラスはC言語のデータ型のようなもので、宣言された変数をインスタンスと呼ぶ

クラスの定義

- クラスの定義は以下の例のようにする

```
class  Message {
private:
  char* message;
public:
  Message();
  ~Message();

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

クラスを宣言するキーワード

クラス名

アクセス制御のためのキーワード：非公開

アクセス制御のためのキーワード：公開

コンストラクタ

デストラクタ

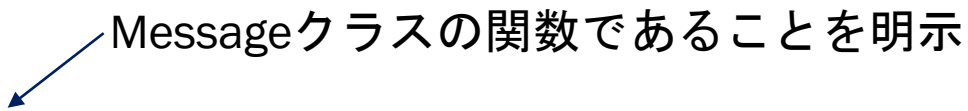
アクセス制限の使い分け

- 基本的にはメンバ変数はprivateに設定して、クラス外から自由に変更できないようにする
  - メンバ変数の値を取得する関数、値を変更する関数をメンバ関数として作成する
  - privateに設定された変数はそのクラスを継承したクラスからも参照できないので、継承するクラスから参照できるようにするために、protectedに設定する。

- メンバ関数は基本的にpublicに設定する

クラスの実装：コンストラクタ

- クラスのインスタンスが作成されたときに自動的に呼び出される関数
- メンバ変数の初期化などを行う
- コンストラクタの名前はクラス名と同じにする




```
Message::Message() {
  message = nullptr;
};
```

コンストラクタ内でのメンバ変数の値の初期化

- コンストラクタ内でメンバ変数の値を初期化する場合、単純な値の代入であれば以下のようにすることができる

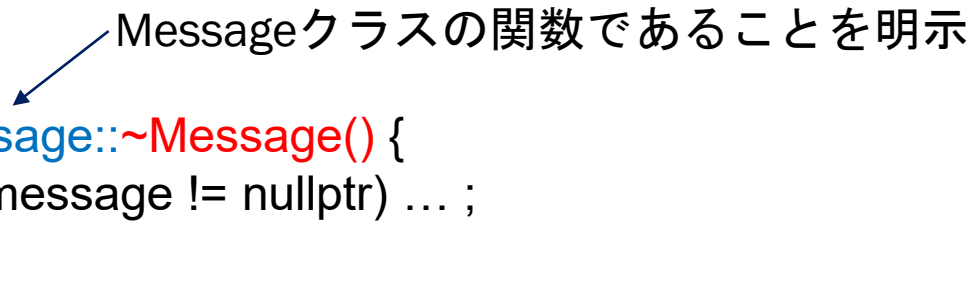
```
Message::Message() {
  message = nullptr;
};
```

```
Message::Message(): message(nullptr) {

};
```

クラスの実装：デストラクタ

- クラスのインスタンスが使われなくなったときに自動的に呼び出される関数
- メンバ変数が動的に領域確保されたメモリ領域を扱っている場合などに、そのメモリ領域を解放するなどの処理を行うための関数
- デストラクタの名前は～クラス名とする



演算子のオーバーロード

- 演算子を独自クラス用に定義しなおすことができる
- クラスのメンバ関数ではない
  - クラス定義の外で演算子のオーバーロードを定義する
  - friend キーワードをつけてクラス定義内で定義することもできる

```
class  Message {
 …
 };
std::istream  &operator>>(std::istream&  stream, Message& obj);
Std::ostream &operator<<(std::ostream& stream, Message& obj);
```

```
class  Message {
 …
  friend std::istream  &operator>>(std::istream&  stream, Message& obj);
  friend std::ostream &operator<<(std::ostream& stream, Message& obj);
};
```

参照呼出し

- 関数間での変数の受け渡し方式には次の２つの方式がある
  - 値呼び出し（call by value)
  - 参照呼出し（call by reference)
- 値呼び出しはC言語で用いられる方式
  - 変数の値をコピーして受け渡しする
  - ポインタ変数を用いた受け渡しも値呼び出しである
- 参照呼出しは呼び出す側の変数と呼び出される側の変数が完全に同一
  - 関数の仮引数の宣言部分で「＆」をつける
  - 呼び出す際には変数名を指定するだけでよい

```
void func1 (void) {
  int a = 0;

  func2 (a);

}
```

```
void func2 (int& b) {
  b = 10;

}
```

クラスの継承

- クラスを定義する際、既存のクラスの機能を継承してクラスを定義することができる
  - 既存クラスの機能を実装し直すことなく、既存クラスにない機能だけ実装するだけで済む
- 継承元のクラスのメンバ変数やメンバ関数を継承先のクラスでも使用することができる

継承元のクラス名

```
class  RepeatMessage: public Message {

private:
  int nloops;

public:
  RepeatMessage();
  ~RepeatMessage();

  …
};
```

```
class  Message {

protected:
  char* message;

 …
};
```

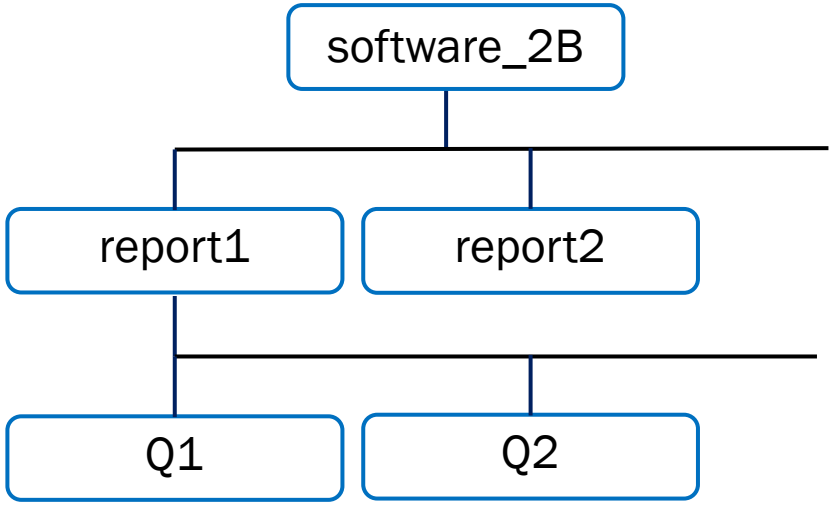
継承するクラス内でメンバ変数にアクセスしたい場合に指定する

小ネタ：CとC++の違い

- ターミナルへの出力
  - C: fprintf (stdout, ”hello world¥n”);
  - C++: std::cout << ”hello world” << std::endl;
- メモリ領域の確保と解放
  - C:
    ```
    int* array = (int *) malloc (sizeof (int));
    free (array);
    int* array = (int *) malloc (sizeof (int) * 10);
    free (array);
    ```
  - C++:
    ```
    int* array = new int;
    delete array;
    int* array = new int [10];
    delete [] array;
    ```
- ヌルポインタ
  - C: int* array = NULL;
  - C++: int* array = nullptr;

演習用のディレクトリ環境の整備

- 本演習用にディレクトリを作成して、更に課題ごとに以下のようにディレクトリを作成する



- 課題ごとにソースコードを管理して、以前の課題のソースコードを上書きなどしないようにする

ソースコードのコンパイル方法

- C++のソースコードをコンパイルするにはg++を使用する
- 複数のソースコードから一つのプログラムを作成する場合には、それらのファイルを同時に指定する
  - g++ Message.cpp main.cpp
- -c オプションを指定することで単独でファイルをコンパイルして、オブジェクトファイルを生成することができる
  - g++ -c Message.cpp
- -o オプションを指定することで生成する実行ファイルの名前を変えることができる
  - 指定しない場合には生成されるファイル名は a.out
  - g++ Message.cpp main.cpp –o Q1-1