[prog2] Programming C++ (C6) Exercise Guide (Ex09)

11/06, Monday 3rd period.

Ex09 について

- クラスのコンストラクタで初期化子を使う
  ⇒ コンストラクタの代入文を簡略化

（今日の講義分）

- 例外処理
  ⇒ try, throw, catch

（後回しになっちゃったけど、期末にも出ているので…）

コンストラクタ復習

TESTクラスの
デフォルトコンスラクタ（引数無し）は
TEST(){ … }

- クラスの実体を宣言したときに1回だけ実行される初期化処理

クラス名の関数(){ … }
（戻り値は書かない）

関数内に、そのクラス変数の初期化処理を自由記述できる

引数を実装すれば、異なる引数を用いた初期化処理も実装できる

```
#include <iostream>
#include <string>

using namespace std;

class TEST{
private: //メンバー変数は隠蔽
  int val;
  string str;
public: //メンバー関数は外部公開
  TEST();//引数無しコンストラクタ（デフォルトコンストラクタ）
  int getv(void);// メンバー変数valの値を返す（ゲッター）
  string getstr(void); // メンバー変数strの値を返す（ゲッター）
  void selfpower(void); // メンバー変数valの値を２乗するだけの関数
};

int main(){
TEST A;

 cout << A.getv() << endl;
 cout << A.getstr() << endl;
 A.selfpower();
 cout << A.getv() << endl;

return 0;
}

TEST::TEST(){//コンストラクタ（オブジェクト生成時に、変数へ代入処理を行うことで初期化）
    val = 5;
    str = "c++";
}
int TEST::getv(void){
    return val;
}
string TEST::getstr(void){
    return str;
}
void TEST::selfpower(void){
    val = val*val;
}
```

初期化子を使う

- もちろん、コンストラクタ内に代入文を書くことで初期化処理は実装できる（Ex06の通り）
- 初期化子と呼ばれるものを使って、簡単に書く方法がある

  ⇒ クラスの継承の書き方となんとなく似てる？

コンストラクタの実体の後ろに

：初期化したい変数（値）

をつける。複数必要な場合はカンマ区切りで

```
28  TEST::TEST(){//コンストラクタ（オブジェクト生成時に、変数へ代入処理を行うことで初期化）
29      val = 5;
30      str = "c++";
31  }
```

初期化子

```
27          //  ↓↓ここに初期化子を追記した
28  TEST::TEST() : val(5), str("c++")   //オブジェクト生成時に、初期化子によってメンバー変数を初期化
29  {
30     cout << "The object of TEST class is generated." << endl;
31  }
```

例外処理

- C言語のときには、何らかの処理に対するエラー処理を実行した関数の戻り値などを用いて自分で実装していた。
- ※ファイルオープン失敗の判定は fopen関数の戻り値から

```
1  fp = fopen("data.in",r);
2  if(fp==NULL){
3   fprintf(stderr,"No such file.\n");
4   exit(1);
5  }
```

- try, throw, catch の3つのキーワードを組み合わせたより

スマートな例外処理の実装が提供されている。

⇒ C++, C#, Java など、異なる言語でも基本的な構造は同じ

例外処理の3ステップ

- try{ … }

  囲まれた領域内に、例外（異常な処理）が検出されないかを監視する

  ⇒　try節の中で、if文を使うなどして自分で検出処理を書く

- throw

  異常を検出したときに、異常発生の通報をする。

  ・文字列を投げる例：　throw “Exception occurred!!”;

  ・数値を投げる例　　throw -1;

- catch(…){ … }

  通報を受け取って、後処理を行う

  ・文字列例外を受け取る例　catch(const *char e){ cout << e << endl; }

  ・数値例外を受け取る例　　catch(int e){ cout << e << endl; }

例外処理

例外が起こりそうな場所を
try節で囲んで監視する

- 例題2の例外処理

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    int val;
6    cout << "Doctor: You MUST NOT eat sweets, or you shall die!" << endl;
7    cout << "Doctor: I'm always checking you!" << endl;
8
9    cout << "Me: I want to eat ..." << endl;
10   cout << "1. Stake" << endl;
11   cout << "2. Salad" << endl;
12   cout << "3. Salmon" << endl;
13   cout << "4. Strawberry cake" << endl;
14   cin >> val; // 4 が入力されると「例外」を投げる
15
16   try {//監視する
17     if(val == 4){ throw "Doctor: WHY DO YOU EAT SWEETS!!!!"; }//通報(文字列を投げる)
18     if(val < 1 || val > 4){ throw val; }//通報(整数値を投げる)
19   }
20   catch (const char *e) {//例外を受け取る(文字列が飛んできた場合のみ実行)
21     cout << e << endl;
22     cout << "You died." << endl;
23   }
24   catch (int val) {//例外を受け取る(整数値が飛んできた場合のみ実行)
25    cout << "error: (" << val << ") No such items." << endl;
26   }
27
28   return 0;
29 }
```

型が対応する例外を受け取ったときだけ、
そのcatchの中が実行される

例外処理

- 例外を見つけたらとにかくcatchしたい場合
- catch(…){ 例外の後処理 } のように 引数部分を(…)とする

⇒ ただし、throwされた値は受け取ることができない

- エラー情報を残して、後処理（判断）は別のところに任せたい場合

- throw;

とだけ書く。（特に関数をまたいだ例外処理をしたいときに）

例外処理

- 要 include <exception>

・「予め定義された例外ハンドラ」もある。マクロのようなもの

以下は一部の例

- **invalid_argument**　（不適切な引数を受け取った）
- **length_error**　（扱えるデータ長を越えた）
- **out_of_range**　（配列の添字が有効範囲外に出た）
- **range_error**　（演算処理中にオーバーフロー発生）
- **bad_alloc**　（メモリ確保などの失敗）

使用例） throw bad_alloc(); //メモリ確保失敗を通知する

- using namespace std; を使っていないときは　std:: も必要