

第2回C++講座

目次

- ・関数ポインタ
- ・クラスとは？
- ・クラスの基礎
- ・メンバ変数
- ・メンバ関数
- ・アクセス修飾子
- ・スタック・キュー
- ・演習

ポイント

- ・クラスの定義 → 新しい型を作ること。ただのデータの集まりではなく、動作も定義できる
- ・メンバ変数 → オブジェクトごとに保持
- ・メンバ関数 → 「呼び出したオブジェクト」のメンバ変数进行操作する
- ・アクセス修飾子 → メンバの参照範囲の制限
- ・変数の代入 → ビットコピー
- ・メンバ関数 → オブジェクトごとに違う値を持つが、同じ処理をする場合に使う
- ・クラス定義

```
class クラス名
{
    アクセス修飾子 :
        メンバ変数;
        メンバ関数;
};
```

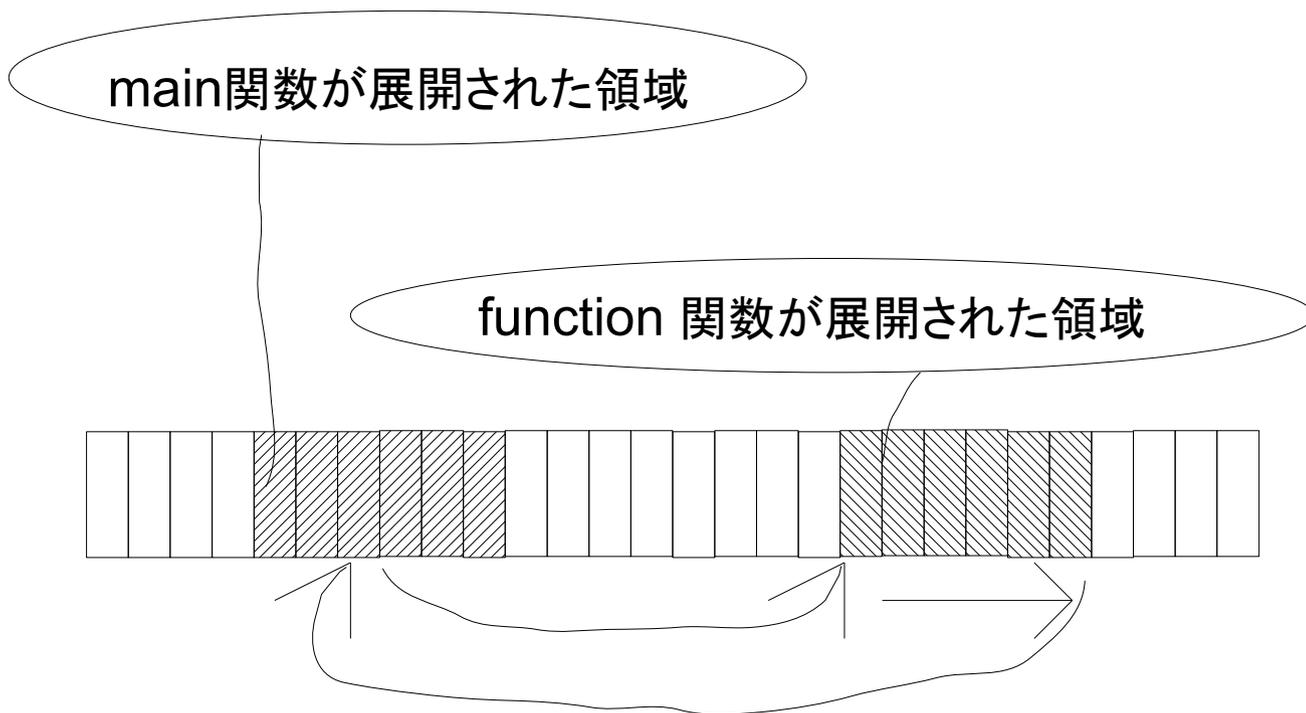
関数ポインタ

関数やプログラムもメモリ上に展開されている
関数の呼び出しはポインタを用いた移動



ポインタを使えば、移動先を
実行中に変えられる！

```
void function()  
{  
    int i = 0;  
    return ;  
}  
  
int main()  
{  
    function();  
    return 0;  
}
```



- ①関数呼び出しでmainからfunctionへ
- ②function実行
- ③mainの呼び出し位置に戻る

関数ポインタ

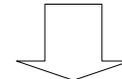
```
#include <iostream>
```

```
struct Enemy{  
    void (*walk)(void);  
};
```

```
void walk1() {std::cout << 1 << std::endl;}  
void walk2() {std::cout << 2 << std::endl;}  
void walk3() {std::cout << 3 << std::endl;}  
}
```

```
int main()  
{  
    Enemy en[3];  
    en[0].walk = walk1;  
    en[1].walk = walk2;  
    en[2].walk = walk3;  
  
    for(int i = 0; i < 3; i++)  
    {  
        en[i].walk();  
    }  
}
```

移動先がどのような関数なのかの情報が必要



戻り値 (*変数名)(引数の型)

例えば、
int* (*func_pointer)(int, int);
は、int型ふたつを引数に持ち、
int* 型を返す関数のポインタ。
名前はfunc_pointer
使うときは普通の関数と同じように使う

```
void function( int (*rand_func)() )  
{  
    int i = rand_func();  
    ....  
}  
  
int main()  
{  
    int (*rand_func)();  
    .....  
    rand_func = rand;  
}
```

関数ポインタ

```
#include <iostream>

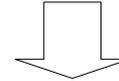
struct Enemy{
    int walk;
};

void walk(int i)
{
    switch(i)
    {
        case 1: std::cout << 1 << std::endl; break;
        case 2: std::cout << 2 << std::endl; break;
        case 3: std::cout << 3 << std::endl; break;
    }
}

int main()
{
    Enemy en[3];
    en[0].walk = 1;
    en[1].walk = 2;
    en[2].walk = 3;

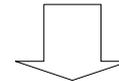
    for(int i = 0; i < 3; i++)
    {
        walk(en[i].walk);
    }
}
```

関数ポインタを使わない場合
switch構文等で代用する



新しい関数を追加するたびに、
関数と数値の対応を覚える必要がある

#defineやenumで対処？



関数を追加するたびに、walkを使用するすべての
コードを再コンパイルする必要が出てくる

(関数ポインタなら初期化部分だけでよい！)
(クラスで同じようなことが可能→普通クラスを使う)

演習：
関数ポインタを使用した例に、
関数を一つ追加して動作を確認する

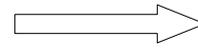
クラスとは？

クラス・構造体の定義 \longrightarrow 新しい”型”を作ること！

(型: プログラム上で使用される変数の種類(ex: int, double))

プログラム上での振る舞いを定義する必要がある

そのクラスの大きさはいくらか？
そのクラスのメモリ構成はどうなっているか？



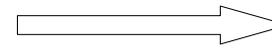
メンバ変数

メンバ変数をどのように操作するか？



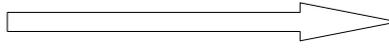
メンバ関数

メンバにどこからアクセスできるか？



アクセス修飾子

初期化時の処理
破棄時の処理はどのようにするか？



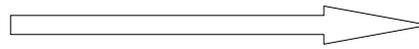
コンストラクタ・デストラクタ
コピーコンストラクタ

代入処理・不等式等はどのように行なうか？



演算子オーバーロード

共通した性質を抜き出す



継承

自分で定義した通りに作用する変数を作ることができる！

クラス定義の書式

```
class クラス名
```

```
{
```

```
    アクセス修飾子 :
```

```
        メンバ変数 ;
```

```
        メンバ関数 ;
```

```
};
```

* C++では、構造体はクラスと同じもの！

class→struct でも同じ

- ポイント

- (1) クラス名、メンバ変数名、メンバ関数名は任意で、クラス名が型名になる
- (2) 要素数、記述の順番は任意
- (3) アクセス修飾子の後ろはコロンのみ
- (4) 構造体同様、定義の最後にはセミコロン
- (5) 変数として使用できるのは、定義が存在する型のみ
- (6) 同一クラスの保持は、定義が終わっていないので出来ない。出来たとしても無限再帰になる
- (7) 同一クラスのポインタは保持できる(ポインタのサイズは決まっている。あるクラスのポインタは、そのクラスの宣言があれば使用できる)

クラスの例

```
//クラス定義
class ClassName
{
public:
    int mem_var1;
    double mem_var2;
    void func();
};

void ClassName::func()
{}

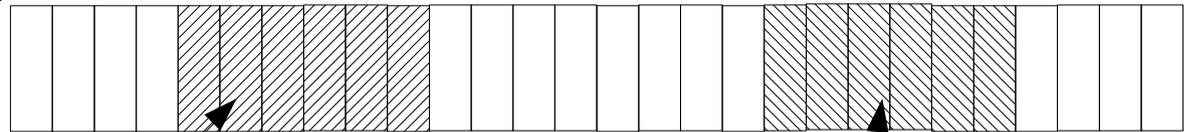
int main()
{
    ClassName obj1, obj2;
    obj1.mem_var1 = 3;
    obj2.mem_var1 = 5;

    return 0;
}
```

クラス定義は、クラスの情報だけを定義しているだけ
メモリ上に実体はまだない

アクセス修飾子は、外部からの参照の可否を決める
public はどこからでも参照可能
その下はメンバ変数とメンバ関数の宣言

メンバ関数の定義。名前空間と似た発想。
どのクラスに属する関数かを::で指定している
グローバル関数と共存可能



ClassName型で定義された領域
obj1専用。int型とdouble型分の領域
こちらのmem_var1に3が代入される

ClassName型で定義された領域
obj2専用。int型とdouble型分の領域
こちらのmem_var1に5が代入される

*赤字は修正箇所。変数名ミス

変数の代入

```
#include <iostream>

class BigData
{
public:
    int bigBit[10000];
};

int main()
{
    BigData source, distance;
    for(int i = 0; i < 10000; ++i)
    {
        source.bigBit[i] = i;
    }
    distance = source;

    std::cout << distance.bigBit[5] << std::endl;

    return 0;
}
```

- 変数の代入では、メンバのビットコピーが行なわれる
- 配列も問題なくコピーされるが、大きいデータのコピーは時間がかかる

演習

1. 矩形を表わすRectangleクラスを作成せよ
この矩形はxy座標上に存在し、x軸、y軸方向に平行であるとして良い
2. bool collide (const Rectangle& lhs, const Rectangle& rhs) で定義される矩形同士の衝突判定関数を作成せよ
3. collide 関数の引数が参照の場合とそうでない場合ではどのような差があるか。また、なぜconst演算子を使用しているか

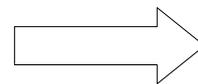
アクセス修飾子

- private ... メンバ関数からのみアクセス可能
- protected ... メンバ関数、派生クラスのメンバ関数からのみアクセス可能
- public ... どこからでもアクセス可能(C言語の構造体のような使い方)
- 何も指定しないとき
 - (1)クラス...指定されていない要素は 全てprivate
 - (2)構造体...指定されていない要素は 全てpublic

クラスを関数のように使用したい

or

クラスをデータの集まりではなく機能として使いたい



どのような変数を持っているかが重要ではなく、変数を操作されたくない

- ・使い方を制限できる。大人数での開発に威力を発揮する
- ・public にする必要がなければ private にするのが基本だが、個人でのプログラムでは比較的どうでもいい

メンバ関数とは

```
#include <iostream>

class MemFunc
{
public:
    int x, y;
    void changeXY(int x, int y)
    {
        this->x = x;
        this->y = y;
    }
};

int main()
{
    MemFunc mf1, mf2;
    mf1.x = 1; mf1.y = 2;
    mf2.x = -1; mf2.y = -2;

    mf1.changeXY(3, 5);

    std::cout << mf1.x << " " << mf1.y << std::endl;
    std::cout << mf2.x << " " << mf2.y << std::endl;

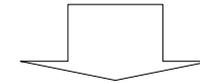
    return 0;
}
```

メンバ変数进行操作するための関数

public, private, protectedすべてのメンバ変数进行操作できる

メンバ関数が操作する変数は、そのオブジェクトが持っているメンバ変数

(左の例では、mf1のx,yだけ変化する)



オブジェクトごとに違う値を持つが、同じ処理をする場合に使う！

thisキーワード(thisポインタ)
メンバ関数内で使用できる。
メンバ関数を呼び出している
オブジェクトのポインタとして扱われる。

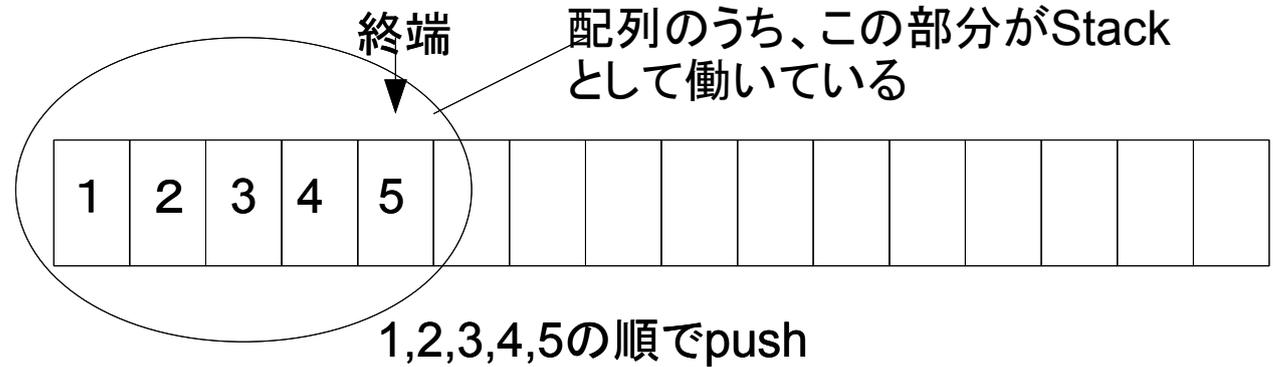
演習

左のchangeXY関数は引数の名前とメンバ変数の名前が衝突している。thisを除くとどうなるか。またthisを使わずに目的を達するにはどうするか

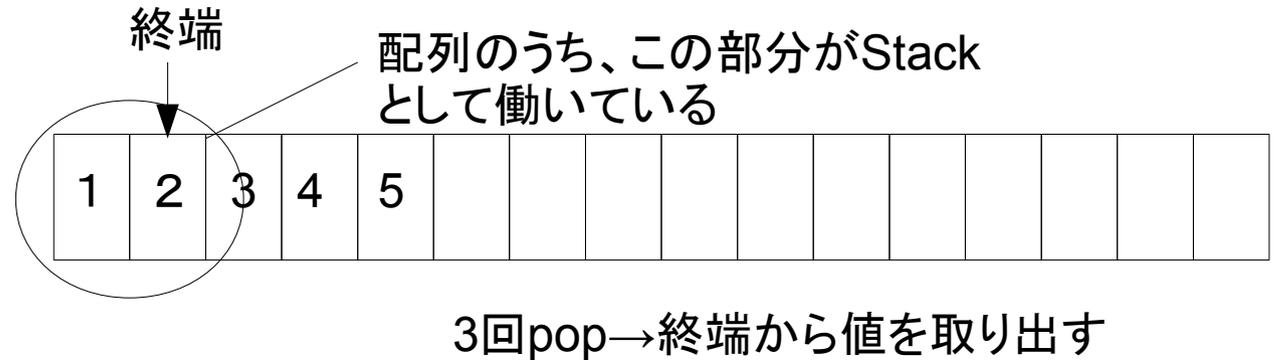
Stack

FILO(First In Last Out)形式のコンテナ

①数値を入れる(push)
Stackの終端を表わす
インデックスを加算していく



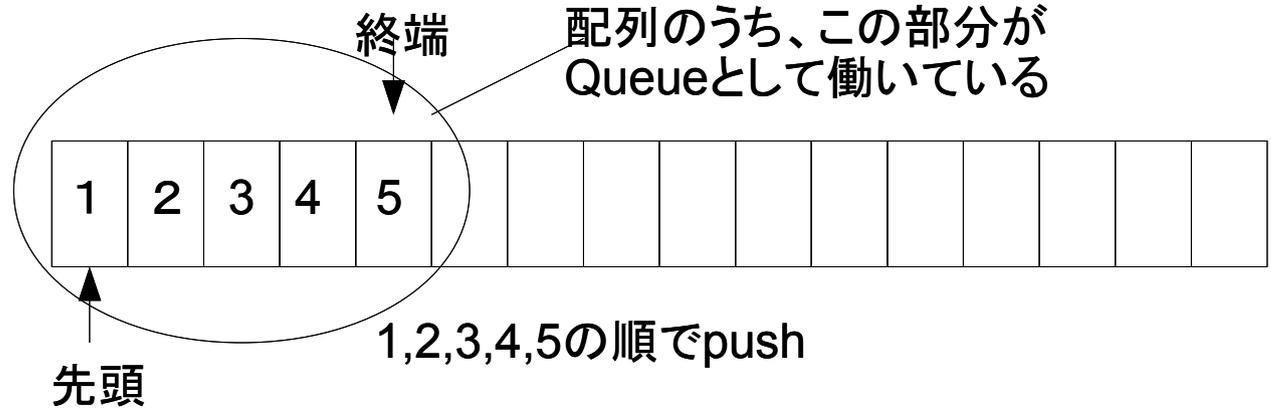
②数値を取り出す(pop)
Stackの終端を表わす
インデックスを減算していく



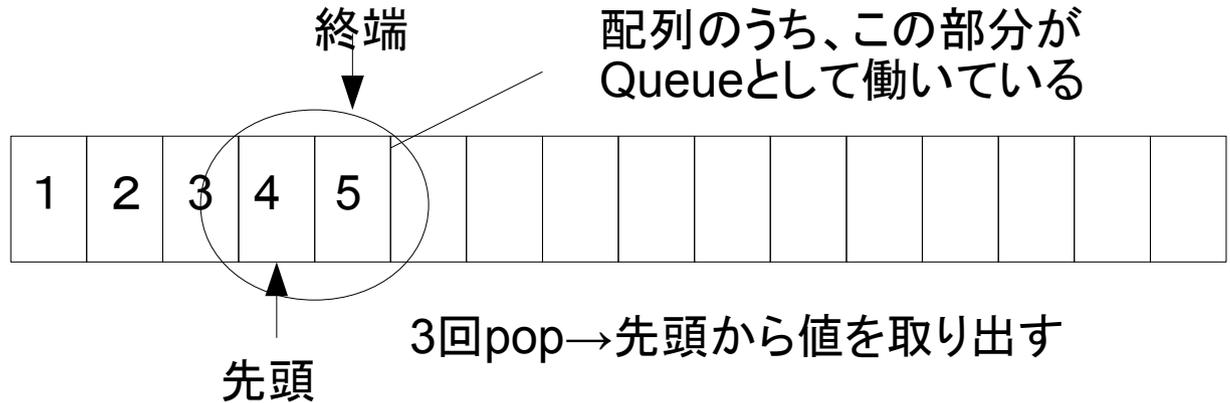
Queue

FIFO(First In First Out)形式のコンテナ

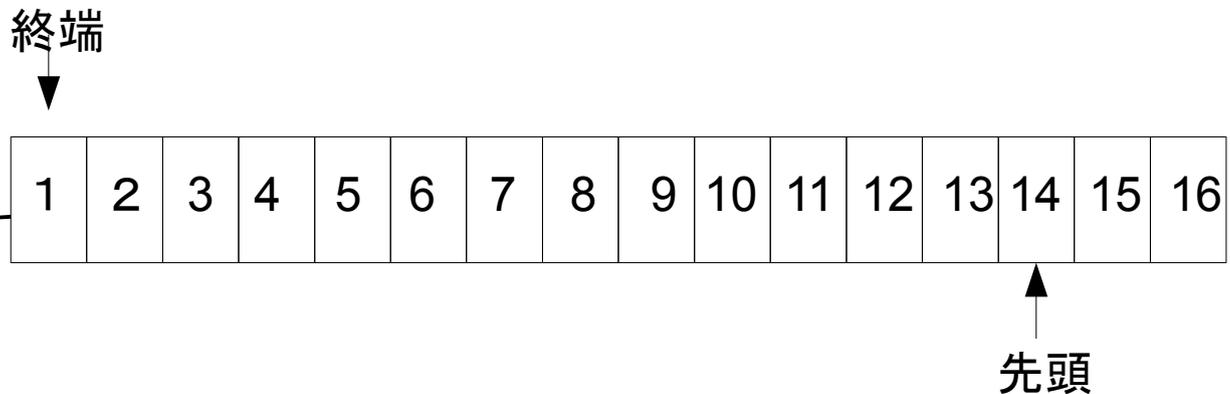
①数値を入れる(push)
Queueの終端を表わす
インデックスを加算していく



②数値を取り出す(pop)
Queueの先頭を表わす
インデックスを加算していく



③配列の端まで来たら?
Queueの終端を表わす
インデックスを配列の先頭に戻す
(%演算子を使う)



演習

char型の整数を、const int MAX_SIZE = 100;まで格納できるキューを作成せよ
ただし、以下のメンバ関数を定義すること

- push 関数…キューに値を入れる。成功したら true を返し、キューがいっぱいならfalse を返す。
- pop 関数…キューから値を取り出す。
- getSize関数…キューに入っている要素の数を返す
- getMax関数…キューに入れられる最大数を返す(マジックナンバーをクラス外で使わないようにするため)
- init関数…キューを初期状態(何も入っていない状態)に戻す

ヒント:メンバ変数→・int型の配列(キュー本体)

- ・配列の使っている部分の先頭・終端のindex
- ・要素数

配列の終端まで来たら先頭に戻る→%演算子

演習

1. 生徒を表わすStudent クラスを作成せよ。名前、ID番号、学年と、このデータが有効かどうかを表わすフラグをメンバ変数として持ち、ID番号が負数の場合にはそのデータは無効であるとする。

標準入力を用いて、このクラスの各要素に代入を行ない、標準出力を用いて結果を表示せよ。データが無効である場合はその旨を表示すること。

2. 前問のStudentクラスについて、データの有効性を表わすフラグが必要でないと感じた。

このクラスからデータの有効性を表わすフラグを削除せよ。また、どのようなクラス構成であれば、他のプログラムの改変を最小限に出来たか考えてみよ。

3. Timer クラスを作成せよ。計測を開始するstart()、計測を終了するend()、計測結果を返すgetResult()、経過時間を返すgetPastTime() の四つの関数をメンバ関数として定義すること。

この際、外部からメンバ変数にアクセスする必要があるか考えよ

* ctime ヘッダの time(0) 関数を使用する