値渡しと参照渡し，ポインタ

ゲームプログラミングA #04

向井 智彦

今週の内容

- 前回課題の振り返り
- 値渡しと参照渡し
  - 講義＆演習
- ポインタ
  - 講義＆演習

先週の演習課題

- Vector3クラスをカプセル化
- Vector3クラスにメンバー関数を追加
  - ベクトル長を求めるLength()
  - ベクトルをゼロクリアするZeroClear()
  - 代入演算「=」
  - ベクトルの加算「+」，減算「-」
  - ベクトルの拡大縮小「*」「/」
  - その他、思いつくものを色々

↑ Vector2クラスの修正例を公開しました

値渡しとは？

C言語にない
C++言語のみの仕様

- 変数が持つデータの内容を，別の変数にコピーして渡す操作
  - コピー先で値が書き換えられても，コピー元には影響を及ばさない
  - 代入演算「=」の振る舞い
  - 関数の引数の振る舞い
  - 関数の戻り値の振る舞い

値渡しの動作確認 (1/2)

```
int main()
{
  int x = 10;
  int y = x;
  x = 5;
  printf(“%d, %d¥n”, x, y); // どんな出力結果？
}
```

値渡しの動作確認 (2/2)

```
void ZeroClear(int a)
{
  a = 0;
}
int main()
{
  int x = 10;
  ZeroClear(x);
  printf(“%d¥n”, x); // どんな出力結果？
}
```

参照とは？

- 同一データに別名を与える処理

```
int main()
{
  int x = 10;
  int &y = x;  // int型変数への参照
  x = 5;
  printf(“%d, %d¥n”, x, y);
  y = 8;
  printf(“%d, %d¥n”, x, y);
}
```

参照渡しとは？

- 参照を通じた関数への引数渡し

```
void ZeroClear(int &a)
{
  a = 0;
}
int main()
{
  int x = 10;
  ZeroClear(x);
  printf(“%d¥n”, x); // どんな出力結果？
}
```

戻り値の代替としての参照渡し

- 関数の出力を受け取るための参照引数

```
void Double(int &output1, int input)
{
  output = input * 2;
}
int main()
{
  int x = 10;
  int y = 0;
  Double(y, x);
  printf(“%d, %d¥n”, x, y);
}
```

クラスの参照渡し

```
void Double(Vector3 &v)
{
  double x2 = 2.0 * v.GetX();
  double y2 = 2.0 * v.GetY();
  double z2 = 2.0 * v.GetZ();
  v.Set(x2, y2, z2);
}

Double(vec);
```

クラスメンバ関数への参照渡し

```
class Vector3
{
public:
  void CopyTo(Vector3 &v) {
    v.Set(x, y, z);
  }
  ...
};

Vector3 a(1.0, 1.0, 1.0);
Vector3 b;
a.CopyTo(b);
```

参照の特徴

- 変数のように後から上書きできない

```
int main()
{
  int x = 10;
  int &y = x;  //yはxの別名
  int z = 0;
  x = 5;
  printf(“%d, %d¥n”, x, y);
  y = z;  //値の代入(≠参照先の変更)
  printf(“%d, %d¥n”, x, y);
}
```

参照を使うケース

- 参照渡し　引数として渡したデータの内容を関数側で書き換えるとき
  - 複数の戻り値→複数の参照渡し
- 巨大なクラスを関数の引数とするとき
  - 値渡しするとコピー/クローンの計算時間が増大
  - 参照渡しだと「別名」を作る処理のみ

演習1

1. 04/sincos_ref.cpp の実装と確認
2. 04/swap_ref.cpp の実装と確認

ポインタ

- 別名の付け方 その2

```
int main()
{
  int x = 10;
  int *y = &x;  //int型変数へのポインタ
  x = 5;
  printf(“%d, %d¥n”, x, *y);
  *y = 8;
  printf(“%d, %d¥n”, x, *y);
}
```

- 注!! 他の情報源では「メモリ」との関連について(正確に)説明されますが本講義では少し触れるに留めます

ポインタを通じた参照渡し

- 参照を通じた関数への引数渡し

```
void ZeroClear(int *a)
{
  *a = 0;
}
int main()
{
  int x = 10;
  ZeroClear(&x);
  printf(“%d¥n”, x); // どんな出力？
}
```

戻り値の代替としてのポインタ参照渡し

- 関数の出力を受け取るための参照引数

```
void Double(int *output, int input)
{
  *output = input * 2;
}
int main()
{
  int x = 10;
  int y = 0;
  Double(&x);
  printf(“%d, %d¥n”, x, y);
}
```

クラスのポインタ渡し

```
void Double(Vector3 *v)
{
  double x2 = 2.0 * v->GetX();
  double y2 = 2.0 * v->GetY();
  double z2 = 2.0 * v->GetZ();
  v->Set(x2, y2, z2);
}
```

v->~~~ の部分は (*v).~~~でもOK

クラスメンバ関数へのポインタ渡し

```
class Vector3
{
public:
    void CopyTo(Vector3 *v) {
        v->Set(x, y, z);
    }
    ...
};

Vector3 a(1.0, 1.0, 1.0);
Vector3 b;
a.CopyTo(&b);
```

ポインタ変数の特徴

- 通常の変数のように後から上書きできる

```
int main()
{
  int x = 10;
  int *y = &x;  //yはxの別名
  int z = 0;
  x = 5;
  printf(“%d, %d¥n”, x, *y);
  *y = &z;  //参照先の変更
  printf(“%d, %d¥n”, x, *y);
}
```

参照渡し vs ポインタ渡し

- 多くの場合，参照渡しで十分

```
void Double(Vector3 &v)
{
  double x2 = 2.0 * v.GetX();
  double y2 = 2.0 * v.GetY();
  double z2 = 2.0 * v.GetZ();
  v.Set(x2, y2, z2);
}
```

```
void Double(Vector3 *v)
{
  double x2 = 2.0 * v->GetX();
  double y2 = 2.0 * v->GetY();
  double z2 = 2.0 * v->GetZ();
  v->Set(x2, y2, z2);
}
```

ポインタを使うケース

- ライブラリ/APIがポインタ使用を想定する場合
- 実行中に参照先を変更する必要がある場合
- プログラム実行中にクラスインスタンスを作る必要があるとき
  - プログラミング中に何個のインスタンスを用意すべきか（≒配列の長さが）わからない場合
  - new/new[] & delete/delete[] ← 後日登場

ポインタの正体 (補足)

- オブジェクトのメモリ位置（アドレス）

- 値渡し
  - オブジェクトのコピー/クローン
- 参照渡し&ポインタ渡し
  - オブジェクトのメモリ位置情報を渡し，その内容を直接読み書き

ポインタの正体(補足)

```
int main()
{
  int value = 0;
  int *ptr = &value;
  int array[5] = {0, 1, 2, 3, 4};
  int *arrayptr = array;
  char str1[6] = “hello”;
  char *str2 = str1;
  printf(“%d, %d, %s¥n”, *ptr, arrayptr[3], str2);
}
```

まとめ

- 参照: 変数に別名を与える
  - 基本型でもクラスでも構造体でも使用可
  - 値渡し ＝ データのコピー/クローンを関数に渡して元データを操作させない
  - 参照渡し ＝ 別名を関数に渡して元データに直接読み書きさせる
- ポインタ: 変数に別名を与える + その他
  - 実体はオブジェクトのメモリ位置
  - 具体的な活用法はゲームプログラミングBにて

演習2

1. 04/sincos_ptr.cpp の実装と確認
2. 04/swap_ptr.cpp の実装と確認

次回以降の予告

- 細かい点の補足（落ち穂拾い）
  - 続・ポインタ
  - thisポインタ
  - コピーコンストラクタ
- ブロック崩しプログラムの解説と改変演習
  - 予習事項：　GitHub Desktop と VisualStudio2017(win) or Xcode(mac)の利用
    - ~~~-GPA/blocks をビルド/コンパイルできるようにする