プログラミング基礎 #12
関数2

担当: 向井 智彦

第7回のまとめ

戻り値の型名　関数名（引数リスト）　{　処理;　}

- 関数
  - 処理のひとまとまり
  - 数値を受け取って（引数）、出力する（戻り値）
- 関数の戻り値と引数
  - 戻り値は1つ以下、引数は0個以上
  - なにも無い場合は void
  - 配列は引数に指定できる、戻り値にはできない
- 関数プロトタイプ宣言とヘッダファイル

本日の内容

- 目標: 関数の利用について理解を深める
- 講義内容:
  - 「関数」のおさらい
  - グラフィクス関数の利用
- 演習: アニメーションの作成

最終課題予告

- これまでに修得したプログラミングスキルを活用したインタラクティブグラフィックス作品
  - テーマは自由
    - 2次元 / 3次元は自由
    - アニメーションしてもしなくてもOK
    - マウス・キーボードの使用/不使用も自由
  - 条件：修得したC++スキルを存分に活用すること
    - 変数、配列、条件分岐、反復 （+αも自由）
- レポート＆作品〆切：１月２９日（火）
  - １月８日＆１５日は制作時間

関数の使い方 (グラフィックス関数)

- 3次元座標指定関数
  - 宣言 `void glVertex3d(double x, double y, double z);`
  - 使い方: `glVertex3d(1.0, 0.0, 3.0);`
- 色指定関数
  - 宣言 `void glColor3d(double r, double g, double b);`
  - 使い方 `glColor3d(1.0, 1.0, 0.0); // 黄`
- 回転関数
  - 宣言 `void glRotated(double a, double x, double y, double z);`
  - 使い方 `glRotated(45.0, 0.0, 0.0, 1.0);`

関数の引数

- glVertex3d(〇〇, ××, △△)
  - の「〇〇」とか「×× 」とか「 △△ 」
- 呼び出し元から，呼び出し先に引き渡す数
  - 例：atan2(1.0, 0.5);
    - 引数1： 1.0 という浮動小数
    - 引数2： 0.5 という浮動小数

$$\tan^{-1}\frac{1.0}{0.5}$$

数学関数の例

```
#include <iostream>
#include <cmath>
using namespace std;
int main(void)
{
    cout << sin(0.52359877) << endl;
    cout << cos(0.52359877) << endl;
    return 0;
}
```

数学関連ヘッダファイル

戻り値

引数

関数名

関数と配列（グラフィックス関数）

- 色指定関数
  - `float teapot1Color[4] = { 1.0, 0.2, 0.2, 1.0 };`
  - `glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, teapot1Color);`
- ライト位置関数
  - `float lightPos[4] = { 0, 80.0, 100.0, 1.0 };`
  - `glLightfv(GL_LIGHT0, GL_POSITION, lightPos);`
- ライト色関数
  - `float lightColor[4] = { 1.0, 1.0, 1.0, 1.0 };`
  - `glLightfv(GL_LIGHT0, GL_DIFFUSE,  lightColor);`
  - `glLightfv(GL_LIGHT0, GL_SPECULAR, lightColor);`

配列と関数

空の大括弧　　配列の要素数

```
double Sum(double a[], int n) {
  double sum = 0.0;
  for (int i = 0; i < n; ++i)
    sum += a[i];
  return sum;
}
int main() {
  double data[4] = {0.0, 1.0, 2.0, 3.0};
  cout << Sum(data, 4) << endl;
}
```

どんな動作？

配列名を指定

#include および ヘッダファイルの正体

- ヘッダファイル＝関数プロトタイプ宣言の羅列
  - sin 関数にはどんな引数がある？
  - 引数の数は？引数の型は？
  - 戻り値はどんな型？
- ライブラリが提供する関数のカタログリスト
  - …のようなもの
  - 中身のコードは知らずとも使うための情報
  - #include することで呼び出し可能に

アニメーション制作に向けて

OpenGLの移動・回転・拡大縮小

```
glPushMatrix();
glTranslated(x移動, y移動, z移動);
glRotated(回転量, 回転軸x, 軸y, 軸z);
glScaled(x拡縮率, y拡縮率, z拡縮率);
glutSolidSphere(10.0, 20, 20);
glPopMatrix();
```

移動・回転・拡大縮小の影響範囲は Push–Pop 内

これ以降の物体を移動

これ以降の物体を回転

これ以降の物体を拡縮

TIPS

・移動→回転→拡縮の順に書く （ほうが直感的）

・各物体毎に Push & Pop で囲む
（囲まないとよくわからない動きになる）

アニメーションと数学

- 四則演算だけでも面白いことはできる、でも
- 周期的に動かす: 正弦 sin / 余弦 cos
  - amp * sin(time * speed + phase);
  - amp: 振れ幅、speed: 振動速さ、phase: 時間差
  - sin & cos の組み合わせ→リサージュ
- 加速する: べき乗 pow / 指数 exp
- 減速する: 対数 log / 指数 exp(負値)
- ジャンプする: 条件分岐 = 論理

ランダムな整数

- int rand(void) （#include <cstdlib> が必要）
  - 0～1.0 の浮動小数への変換
    - double rnd = double(rand()) / RAND_MAX;
  - -100～100の浮動小数への変換
    - double rnd = double(rand()) / RAND_MAX * 200.0 – 100.0
  - 0～9の整数型乱数への変換
    - int rnd = rand() % 10;
  - -100～100の整数型乱数への変換
    - int rnd = rand() % 201 – 100;

関数を活用するメリット

- プログラムを部品化 （モジュール化）
  - よく使う処理をまとめておく
  - バグを減らす，読みやすくする
  - 開発作業量の最小化
  - 過去の資産の使い回し
- 他者の力を借りる・他者に力を借す
  - ライブラリ，API，ミドルウェア等の機能を呼び出して活用可能

長いプログラムを書くときのTIPS

- こまめにcommit＆pushしておく
  - githubが全ての作業履歴を記録する
- コメントを書いておく(// あるいは /* 文 */)
  - 検索キーワードだけでも書いておく(Ctrl + F3)
- コードを整形しておく
  - インデントや空白に一貫性を持たせる
- 関数化:意味単位にまとめる
  - プログラムが読みやすくなる→色々な苦労が減る