

プログラミング基礎 #06

関数

担当：向井 智彦

前回のおさらい

- for (初期化式; 継続条件式; 更新式) { }
- while (継続条件式) { }
- 反復の中断には break
- 反復対象ブロック内の後略には continue
- 変数のスコープは宣言した中括弧 { } 内
- スコープに対応したインデントで整形を
 - ついでに単語・演算子の前後の空白も

本日の内容

- 関数の使い方と作成
- 関数と配列
- ヘッダファイル
- ライブラリ

@ wandbox.org

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

文法図解

```
#include <iostream>
#include <cmath> 数学関連ヘッダファイル
using namespace std;
戻り値 int main(void)
{
    関数名 sin(引数 0.52359877) << endl;
    cout << cos(0.52359877) << endl;
    return 0;
}
```

The diagram illustrates the grammar of the provided C++ code. It uses arrows to link Japanese labels to specific tokens or expressions in the code:

- 戻り値** (Return value) points to the `int` type specifier in the `main` function signature.
- 関数名** (Function name) points to the `main` identifier.
- 引数** (Argument) points to the `void` parameter in the `main` function signature.
- 引数** (Argument) points to the numerical value `0.52359877` in both `sin` and `cos` function calls.

main関数, 使ってますか？

```
int main(void)
{
    cout << "Hello world!" << endl;
    return 0;
}
```

これって何？

main関数 = OSが呼び出す関数

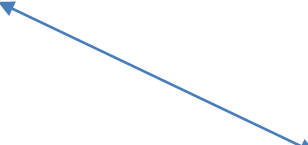
1. Windows エクスプローラーで xxx.exe を起動
2. Windows が、xxx.exe の中にある **main** 関数を探し出す.
3. 見つかったmain関数をWindowsが**呼び出す**
4. main関数内の処理("Hello world")を実行
5. main関数から 0 という**戻り値**がwindowsへ
戻り, xxx.exeは正常終了

関数とは？

- 別の関数から呼ばれて,
 - Windows calls main function
- 何かしらの処理を実行し,
 - `cout << "hello world" << endl;`
- 呼び出し元に何らかの値を戻す,
 - `return 0;`
- 処理のひとまとまり

何もしないmain関数の作り方

```
int main(void)
{
    return 0;
}
```



- int: 戻り値の型
- main: 関数の名前
- void: 引数なし
- return 0;
 - 値を返してmain関数を終了

関数の基本形

戻り値の型名 main(void) { return 0; }



戻り値の型名 関数名 (引数リスト) { 処理; }

関数とは？

- 別の関数から呼ばれて,
 - Windows calls main function
- 何かしらの処理を実行し,
 - `cout << "hello world" << endl;`
- 呼び出し元に何らかの値を返す, ←何型？
 - `return 0;`
- 処理のひとまとまり ←名前は何？

関数の引数

- 名前(〇〇, × ×)
 - の「〇〇」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

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

関数の基本形

戻り値の型名 `main(void) { return 0; }`



戻り値の型名 関数名 (引数リスト) { 処理; }


戻り値と引数 補足

戻り値の型名 関数名 (引数リスト) { 処理; }

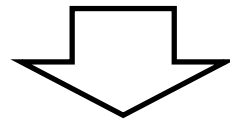
- 戻り値は1つのみ (2つ以上戻す方法は来週)
 - `int function() { return x; }`
- 戻り値がない場合は `void`
 - `void function() { ... };`
- 引数は0個以上の任意の数
 - `int Sum(int v0, int v1, ..., int v99) { ...; return sum; }`
- 引数がない場合は `void`
 - `void Product(int v0, ..., int v99) { ... }`

関数の使い方 (数学の例)

- 絶対値の和を求める

$[a_{11} \quad a_{21} \quad a_{31}]$  配列
`double a[3];`

$a[0]$ の絶対値 + $a[1]$ の絶対値 + $a[2]$ の絶対値

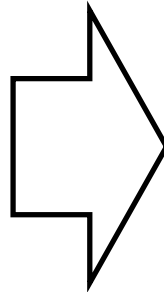


```
double sum = 0;
for (int i = 0; i < 3; ++i) {
    sum += 絶対値を求める関数( a[i] );
}
```

関数作成法：mainを修正

戻り値の型名 関数名 (引数リスト) { 処理; }

```
int main(void)
{
    cout << "Hello!";
    return 0;
}
```



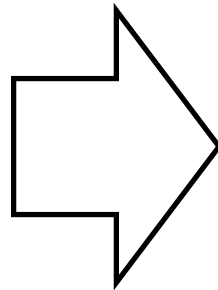
```
int Hello(void)
{
    cout << "Hello!";
    return 0;
}
```

Hello関数完成

Helloを呼び出す

戻り値の型名 関数名 (引数リスト) { 処理; }

```
int Hello(void)
{
    cout << "Hello!";
    return 0;
}
```



main関数を
追加し、
Hello関数を
呼び出す

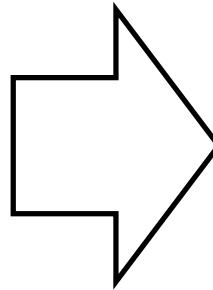
```
int Hello(void)
{
    cout << "Hello!";
    return 0;
}
int main()
{
    Hello();
    return 0;
}
```

引数を加える

戻り値の型名 関数名 (引数リスト) { 処理; }

戻り値は不要→void

```
void Hello(void)
{
    cout << "Hello!";
    return 0;
}
int main()
{
    Hello();
    return 0;
}
```



関数名を修正

```
void Print(int hiki)
{
    cout << hiki;
}

int main()
{
    int x = 5;
    Print(x);
    return 0;
}
```

変数の持つ
値を渡す

来週の予告：関数とスコープ

戻り値の型名 関数名 (引数リスト) { 処理; }

```
void ClearPrint(int hiki) {  
    hiki = 0;  
    cout << hiki << endl;  
}  
  
int main() {  
    int hiki = 10;  
    ClearPrint(hiki);  
    cout << hiki << endl;  
    return 0;  
}
```

変数宣言の重複

```
#include <iostream>  
int main(void) {  
    int sum = 0;  
    for (int i = 0; i < 10; i = i + 1) {  
        int sum = 0;  
        sum = sum + i;  
    }  
    std::cout << sum << std::endl;  
}
```

青sumと赤sumは別物

赤sumのスコープが優先

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;  
}
```

どんな動作？

```
int main() {  
    double data[4] = {0.0, 1.0, 2.0, 3.0};  
    cout << Sum(data, 4) << endl;  
}
```

配列名を指定

配列と関数 contd.

```
void Clear(double a[], int n) {  
    for (int i = 0; i < n; ++i)  
        a[i] = 0.0;  
}
```

どんな動作？

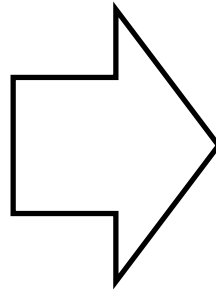
```
int main() {  
    double data[3] = {5.0, 4.0, 3.0};  
    Clear(data);  
    cout << data[0] << endl;  
    cout << data[1] << endl;  
    cout << data[2] << endl;  
}
```

配列と関数：できないこと

```
int[] CreateArray() {  
    int a[3] = {0, 0, 0};  
    return a; // 配列全体は戻せない  
}  
  
int main() {  
    double data[] = CreateArray(data); //エラー  
    cout << data[0] << endl;  
    cout << data[1] << endl;  
    cout << data[2] << endl;  
}
```

少し実験

```
void Print(int hiki)
{
    cout << hiki;
}
int main()
{
    Print(5);
    return 0;
}
```

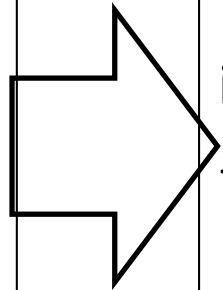


main関数と
Print関数の
順番を
入れ替える

```
int main()
{
    Print(5);
    return 0;
}
int Hello(int hiki)
{
    cout << hiki;
}
```

プロトタイプ宣言

```
void Print(int hiki)
{
    cout << hiki;
}
int main()
{
    Print(5);
    return 0;
}
```



```
void Print(int hiki);
```

```
int main()
```

```
{
```

```
    Print(5);
```

```
    return 0;
```

```
}
```

```
int Print(int hiki)
```

```
{
```

```
    cout << hiki;
```

```
}
```

↑
{ 処理; } が無い点に注目
その代わりに「;」がある

{ 処理; } がある以外は
プロトタイプ宣言と一緒に



言葉は難しいけど、今までやっていることと一緒に
変数の宣言 ⇔ 関数のプロトタイプ宣言

#include および ヘッダファイルの正体

- ライブラリはどんな関数を提供している？
 - カタログリストのようなもの
- 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？
 - 中身のコードは知らずとも使うための情報

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呼び出して活用可能

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