C言語の基礎4

関数

関数の定義

関数の書式

```
戻り値の型 関数名（引数の型 引数1,引数の型 引数2,・・・,引数の型 引数n）
{
  処理
  return 戻り値;
}
```

- 戻り値の型が「`void`（無し）」の場合は、`return`文は省略可

関数の例1

- 2数の和を求める関数`add`の定義（func_1.c）

```c
#include <stdio.h>

int add(int a, int b) {
    return a + b;
}

int main(int argc, char const *argv[]) {
    int x = 10, y = 20;
    int z;

    z = add(x, y);
    printf("%d + %d = %d\n", x, y, z);
    return 0;
}
```

- `warning`になる書き方（func_2.c）
    - コンパイルは完了し、`a.out`は一応、動作する

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {
```

```
    int x = 10, y = 20;
    int z;

    z = add(x, y);
    printf("%d + %d = %d\n", x, y, z);
    return 0;
}

int add(int a, int b) {
    return a + b;
}
```

- `Error` になる書き方（func_3.c）
    - `a.out` の生成が出来ない

```
#include <stdio.h>

int main(int argc, char const *argv[]) {
    int x = 10, y = 20;
    int z;

    z = add(x, y);
    printf("%d + %d = %d\n", x, y, z);
    return 0;
}

float add(int a, int b) {
    return a + b;
}
```

プロトタイプ宣言

- 関数を定義する前に、使用する場合は、プロトタイプ宣言（事前に関数の型を明確にする）を行う。
    - プロトタイプ宣言がない場合、使用した関数はint型変数を渡し、int型を返すものとしてコンパイル処理を行う。

      ```
      int func_x(int);
      ```

      とみなした上でコンパイルを行い、実際の関数が出てきたときに、型が異なるとエラーが発生する。

【func_2_2.c】

```c
#include <stdio.h>

int add(int, int);  // プロトタイプ宣言

int main(int argc, char const *argv[]) {
    int x = 10, y = 20;
    int z;

    z = add(x, y);
    printf("%d + %d = %d\n", x, y, z);
    return 0;
}

int add(int a, int b) {
    return a + b;
}
```

【func_3_2.c】

```c
#include <stdio.h>

float add(int a, int b);  // このように書いても良い

int main(int argc, char const *argv[]) {
    int x = 10, y = 20;
    int z;

    z = add(x, y);
    printf("%d + %d = %d\n", x, y, z);
    return 0;
}

float add(int a, int b) {
```

```
    return a + b;
}
```

- この場合 `add` 関数は `float` で答えを返すことが、コンパイル時に判断できるため、適切にコンパイルが可能になる。
- `z = add(x, y);` で型変換が行なわれる。
    - `z` の `int` 型に変換後格納される。

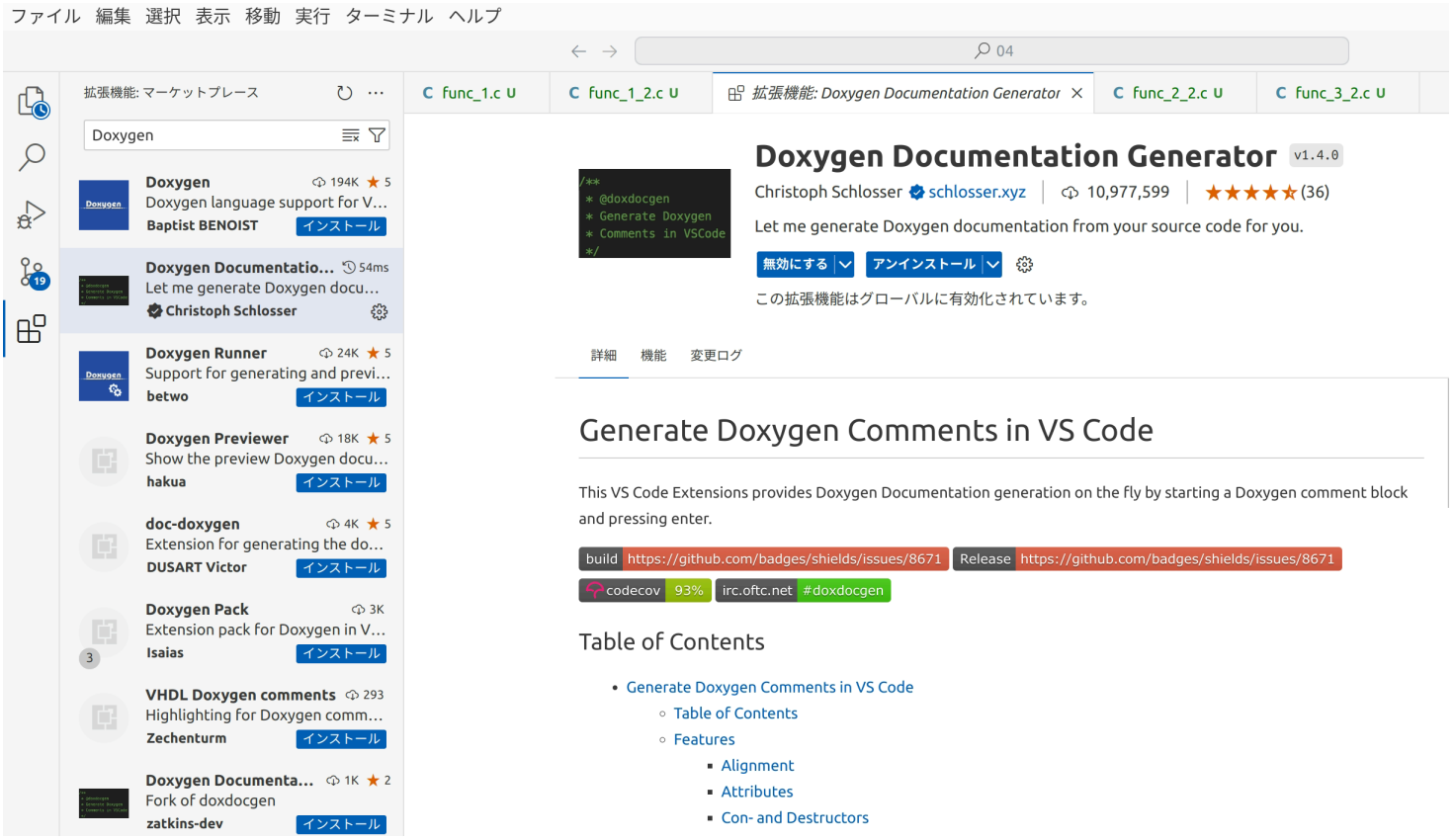
プロトタイプ宣言は書くべき

- 使用順序により記述する順序を考えるのは、規模が大きくなると難しくなる（func_4.c）
    - 以下の例はコンパイルに失敗する。

```
#include <stdio.h>

int fa(int a, int b) {
    return a + fx(b);
}

float fx(int a) {
    return a * 1.0;
}

int main(int argc, char const *argv[]) {
    int x = 10, y = 20;
    int z;

    z = fa(x, y);
    printf("%d + %d = %d\n", x, y, z);
    return 0;
}
```

コメントの入れ方（Doxygen）

- C , C++ , Python, PHP, Java, C#, Objective-C, Fortran, VHDL, Splice, IDL, and Lex などで利用されるツール＆記述方法
- 使い方は専用ツールを使うのが一般的だが、vscode上では以下の拡張機能を使用することで最低限の利用が可能



- 使い方
  - 作成した関数の上の行で、`/**[Enter]`と入力することで、雛形を作成してくれる。
    - 最低限`@breif`項目と、`@param`, `@return`だけは確認しておく。

【func_1_2.c】

```c
#include <stdio.h>

/**
 * @brief 2つの整数の和を返す
 *
 * @param a
 * @param b
 * @return int
 */
int add(int a, int b) {
    return a + b;
}
```

練習

- 以下のプログラムを作成してください。
  ただし、関数だけでなく作成した関数を呼び出すmainも作成し、動作するようにしてください。

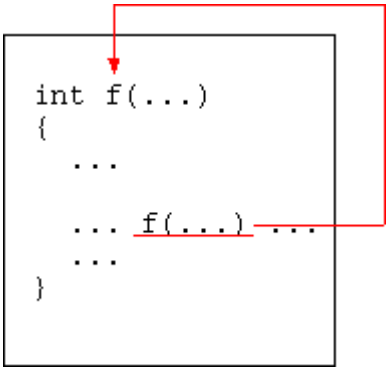
- 一つの整数を引数として受け取り、その数が偶数か奇数かを判定して、偶数の場合は1、奇数の場合は0を返す関数 `is_even` を作成してください。（prog1.c）
    - true / falseでも可
- 一つの整数を引数として受け取り、その数以下の自然数の和を返す関数 `sum` を作成してください。（prog2.c）
- 一つの整数を引数として受け取り、その数が素数かどうかを判定して、素数の場合は1、素数でない場合は0を返す関数 `is_prime` を作成してください。（prog3.c）
    - true / falseでも可
- 一つの整数を引数として受け取り、その数の桁数を求めて返す関数 `count_digits` を作成してください。（prog4.c)

再帰呼出

- 自分自身を呼び出す関数

> https://ylb.jp/2006b/proc/recursion/

ある関数 `f` の定義の中に `f` 自身を呼び出している箇所があるとき、その呼び出しを再帰呼出し(recursive call)と言います。また、そのような定義は再帰的(recursive)である、と言います。



- メリット
  プログラムがわかりやすくなる場合がある
- デメリット
  メモリ使用量が増える場合がある
  （スタックオーバーフローが起こる場合がある）
  処理時間が増える場合がある

- 以前作成した、階乗を求めるプログラムを思い出してほしい（以下プログラム例）

【02_prog3.c】

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {
    int in;
    long res = 1;

    printf("求めたい数字を入力してください：");
    scanf("%d", &in);

    for (int i = 1; i <= in; i++) {
        res *= i;
    }

    printf("%dの階乗は%ldです\n", in, res);
    return 0;
}
```

- これを関数を使用する形に書き換える

【03_prog3.c】

```c
#include <stdio.h>

// プロトタイプ宣言
long factorial(int);

// 関数定義
long factorial(int n) {
    long res = 1;
    for (int i = 1; i <= n; i++) {
        res *= i;
    }
    return res;
}

int main(int argc, char const *argv[]) {
    int in;

    printf("求めたい数字を入力してください：");
    scanf("%d", &in);

    printf("%dの階乗は%ldです\n", in, factorial(in));
    return 0;
}
```

- 関数 `factorial` を再帰呼び出しに書き換える

【03_prog3r.c】

```c
#include <stdio.h>

long factorial(int);

/*
    階乗を求める数学的な定義
    n! = n * (n-1)! ：n >= 1
    0! = 1        ：n = 0
*/
long factorial(int n) {
    if (n == 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

```c
int main(int argc, char const *argv[]) {
    int in;

    printf("求めたい数字を入力してください：");
    scanf("%d", &in);

    printf("%dの階乗は%ldです\n", in, factorial(in));
    return 0;
}
```

- このプログラムには、問題点がある。
    - どのような問題があるか？
    - 出力されるエラーの意味は？
    - 正しく動作するように、修正してください。

練習

- 次のプログラムは、どのような処理を行なっているプログラムか？
    - 実際に動かし、変数num1,num2の値を変更して確認してください。
    - 関数gcdはどのようなアルゴリズムで実装されていますか？
      一般的な名称を調べてください。
    - 関数gcdを再帰呼び出しを使って書き換えてください。（prog5r.c）
      正しく動作するのを確認してください。

【prog5.c】

```c
#include <stdio.h>

int gcd(int, int);

int gcd(int a, int b) {
    while (b != 0) {
        int remainder = a % b;
        a = b;
        b = remainder;
    }
    return a;
}

int main() {
    int num;
    int num1 = 12;
    int num2 = 18;

    int result = gcd(num1, num2);
```

```
    printf("The GCD of %d and %d is %d\n", num1, num2, result);
    return 0;
}
```

- 次のプログラムの実行結果を考えてください。(prog6.c)
    - 机上トレース（PCを使用しないでプログラムをデバッグ・トレース）してください。

```c
#include <stdio.h>

void recfunc(int);

int main(int argc, char const *argv[]) {
    recfunc(4);
    return 0;
}

void recfunc(int n) {
    if (n == 0) {
        return;
    } else {
        printf("%d\n", n);
        recfunc(n - 1);
        printf("%d\n", n);
        return;
    }
}
```

練習

- 以下のプログラムを再帰呼び出しを使用して作成してください。

1. 与えられた整数の逆数の和を再帰的に計算する関数 `reciprocal()` (prog7.c)
   【実行結果】

```
$ ./a.out
整数nを入力してください：5
5までの逆数の和は2.283334です。
```

   【考え方】

   - Nまでの逆数の和

$$\frac{1}{N}+\frac{1}{N-1}+\frac{1}{N-2}+\cdots+\frac{1}{1}$$

2. 与えられた数の二進数表現を再帰的に生成する関数 `binary()` (prog8.c)
   【実行結果】

```
$ ./a.out
整数nを入力してください：10
10の二進数表現は： 1010
```