

第5回C言語勉強会

関数

機能に名前を与える

- C言語が言語の仕様として持っている、原始的で小さな部品を組み合わせることで、様々な機能を実現することができます。
- そのような機能に名前を付け、抽象的に扱えるようにすると、コードの可読性が向上します。
- それを実現するのが、関数です。

関数

- いままでも、printf, scanfなどの関数を使ってきました。それぞれ表示する、読み込むなどの機能を持っていましたね。

関数の基本

- 関数は、引数(ひきすう)をとり、それに対して動作を起こします。printfを例にとると、

```
0 1 2 3
1 #include <stdio.h>↓
2 ↓
3 int main(void) ↓
4 { ↓
5     printf("Hello, world¥n"); ↓
6     return 0; ↓
7 } [EOF]
```

```
C:\¥windows¥system32¥cmd.exe
Hello, world 動作
続行するには何かキーを押してください
```

関数を定義する

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int limit=10;
6
7     int i;
8     for (i=0; i<=limit; i++)
9     {
10        printf("%2d * %2d = %3d¥n", i, i, i*i);
11    }
12
13    return 0;
14 } [EOF]
```

この動作を関数として使いたいとすると

```
0 1 2 3 4
1 #include <stdio.h>
2
3 void print_iter(int); ①
4
5 int main()
6 {
7     int limit=10;
8
9     print_iter(limit); ②
10
11     return 0;
12 }
13
14 void print_iter(int limit) ③
15 {
16     int i;
17     for(i=0; i<=limit; i++)
18     {
19         printf("%2d * %2d = %3d¥n", i, i, i*i);
20     }
21 } [EOF]
```

左の図のように①、③を用意することで、main()のなかで②が使えるようになります。

①: プロトタイプ宣言

```
3 void print_iter (int);
```

戻り値の型 関数名 引数の型

The diagram shows a C function prototype declaration: `void print_iter (int);`. The word `void` is underlined in blue, with a blue line pointing to the label '戻り値の型' (Return type). The words `print_iter` are underlined in red, with a red line pointing to the label '関数名' (Function name). The word `int` is underlined in orange, with an orange line pointing to the label '引数の型' (Argument type). The entire code line is highlighted in a light yellow background.

関数名 : 文字どおり関数の名前

引数の型 : その関数の取る引数の型を指定してやる

戻り値の型 : 後程解説します。

③: 関数定義

```
14 void print_iter (int limit) ←  
15 { ←  
16     int i; ←  
17     for (i=0; i<=limit; i++) ←  
18     { ←  
19         printf ("%2d * %2d = %3d¥n", i, i, i*i); ←  
20     } ←  
21 } [EOF]
```

関数が受け取った引数の、関数定義内での名前

関数定義内では、main関数内で書いたようにコードが書けます。

②: 関数呼び出し

- プロトタイプ宣言と、関数定義がきちんとされた関数は、main関数内で使用することができます。

やってみよう

- 先ほどの例で定義した`print_iter()`関数を`main`関数の中で何回か、異なる引数を与えて使ってみよう。
- `print_iter()`関数の定義を変更して、三乗など、いろいろな物を表示させてみよう。

問題1

- 受け取った数の絶対値を表示するコードを書いてください。その中で、引数として数を受け取り、その数の絶対値を表示する関数“print_abs”を定義してください。
- ヒント: まずこの機能を実現するコードがmain関数内でどう書かれるか考えてください(実際に書いてもいいです)。それから、print_absを文法に沿って定義しましょう。

問題1の答え

```
1 #include <stdio.h>↓
2 ↓
3 void print_abs(double);←
4 ↓
5 int main(void)↓
6 {↓
7     double user;←
8     ←
9     printf("数値を入力してください。 :");←
10    scanf("%lf",&user);←
11    print_abs(user);←
12    ←
13    return 0;↓
14 }←
15 ←
16 void print_abs(double n)←
17 {←
18     double ans = n;←
19     if(n < 0)←
20     { ←
21         ans = -1*n;←
22     }←
23     ←
24     printf("%.11f の絶対値 : %.11f\n", n,ans);←
25     ←
26 }[EOF]
```

戻り値

- 問題1で定義してもらった”print_abs”ですが、なんと絶対値を表示するだけで、計算に使う事ができません。
- これを計算には、戻り値を設定してやる必要があります。
- プロトタイプ宣言に戻り値の型、関数定義にreturnを追加することで使えるようになります。

プロトタイプ宣言

```
3 double abs (double) ; ←  
4 ↓
```

戻り値の型を指定する。

戻り値の型の宣言を、プロトタイプ宣言の先頭に追加します。

return

```
10 double abs(double n) ←  
11 { ←  
12     if (n < 0) ←  
13     { ←  
14         return -1 * n; ←  
15     } ←  
16     else ←  
17     { ←  
18         return n; ←  
19     } ←  
20 } [EOF]
```

return で返せるのは、数値、変数など。

複数のreturnを定義に含めることもできる。

戻り値の利用

```
1 #include <stdio.h> ↓
2 ↓
3 int main(void) ↓
4 { ←
5     int string; ↓
6     string = printf("Hello, world¥n"); ←
7     printf("上の文の文字の数は%dです。¥n", string); ←
8     ↓
9     return 0; ↓
10 }|[EOF]
```

関数が戻り値に置き換えられる
ような感じで利用できます。

やってみよう

- 問題1で書いたコードを、abs関数の戻り値を利用する形で書き直してみよう。
- abs関数の定義を工夫して、return文を一つしか使わない形にしてみよう。

```
0 1 2 3 4 5
1 #include <stdio.h>↓
2 ↓
3 double abs(double);←
4 ↓
5 int main(void)↓
6 {←
7   double user;←
8   ↓
9   printf("値を入力してください:");←
10  scanf("%lf", &user);←
11  printf("%.1lf の絶対値: %.1lf¥n", user, abs(user));←
12  ↓←
13  return 0;←
14  ↓
15 }←
16 ←
17 double abs(double n)←
18 {←
19   double ans = n;←
20   if(n < 0){←
21     ans = -1*n;←
22   }←
23   ↓←
24   return ans;←
25 } [EOF]
```

こぼれ話

- 広義の関数: 言語に備わる関数を定義する方法を使って定義されたもの。
- 狭義の関数: 引数を取り、戻り値を返すことしかしない。この定義では、printfなどは関数に含まれない。数学の関数の定義に近い。