目次

25. 関数ポインタ

関数も変数と同じようにアドレスを持ち、関数のアドレスを変数に格納することができます。
今までのように関数名ではなく、関数のアドレスを格納した変数から関数を呼び出すことができます。
関数のアドレスを格納する変数のことを「関数ポインタ」と呼びます。

関数ポインタの使用例

function 関数の関数ポインタを格納するための変数 pFunc は以下のように定義します。

関数の定義

function 関数の定義を以下に示します。

```
void function(int);     /* 復帰値の型 関数名 引数の型 */
```

関数ポインタの定義

ポインタ変数 pFunc の定義を以下に示します。

```
void (*pFunc)(int);     /* 復帰値の型 変数名 引数の型 */
```

ポインタ変数 pFunc を使用して function 関数を呼び出す場合は、以下のようにします。

(1) 関数ポインタに、関数のアドレスを代入

```
pFunc = function;
```

(2) 関数ポインタに入っているアドレスにある関数を実行

```
pFunc(1);
(*pFunc)(1);
```

関数ポインタの利用において、上記の2通りの実行方法があります。

変数のポインタ変数同様、関数ポインタ変数のサイズも 4 バイトです。

関数ポインタは配列に置いたり、関数への引数、復帰値として使用することも可能です。

関数ポインタを用いたプログラム

以下は入力した 2 つの数値に対して四則演算を行うプログラムです。

コード

```
#include <stdio.h>

void func_add(int iNum1, int iNum2)
{
  /* 引数 iNum1 と iNum2 の加算結果を出力する */
  printf("iNum1+iNum2=%d\n", iNum1 + iNum2);
}
void func_sub(int iNum1, int iNum2)
{
  /* 引数 iNum1 と iNum2 の減算結果を出力する */
  printf("iNum1-iNum2=%d\n", iNum1 - iNum2);
}
void func_mul(int iNum1, int iNum2)
{
  /* 引数 iNum1 と iNum2 の乗算結果を出力する */
  printf("iNum1*iNum2=%d\n", iNum1 * iNum2);
}
void func_div(int iNum1, int iNum2)
{
  if (0 == iNum2) {
    /* 引数 iNum2 が 0 の場合はエラー */
    printf("Error.\n");
  }
  else {
    /* 引数 iNum1 と iNum2 の除算結果を出力する */
    printf("iNum1/iNum2=%d\n", iNum1 / iNum2);
  }
}

int main(void)
{
  int  iInput1;
  int  iInput2;
  int  iExpr;

  printf("数値を入力(1回目)");
  scanf("%d", &iInput1);

  printf("数値を入力(2回目)");
  scanf("%d", &iInput2);

  printf("0:加算 1:減算 2:乗算 3:除算\n");
  printf("計算方法を入力 ");
  scanf("%d", &iExpr);

  switch (iExpr) {
  case 0:
    func_add(iInput1, iInput2);
    break;
  case 1:
    func_sub(iInput1, iInput2);
    break;
  case 2:
    func_mul(iInput1, iInput2);
    break;
  case 3:
    func_div(iInput1, iInput2);
    break;
  default:
    /* 範囲外の入力が行われた */
    break;
  }
  return 0;
}
```

結果(例 1)

以下の入力を行なった場合の結果を示します。

1. 1 つ目の数値として 125 を入力
2. 2 つ目の数値として 92 を入力
3. 計算方法として 0 を入力(加算)

結果は以下のようになります。

```
数値を入力(1回目)125
数値を入力(2回目)92
0:加算 1:減算 2:乗算 3:除算
計算方法を入力 0
iNum1+iNum2=217
```

結果(例 2)

以下の入力を行なった場合の結果を示します。

1. 1 つ目の数値として 12 を入力
2. 2 つ目の数値として 20 を入力
3. 計算方法として 2 を入力(乗算)

結果は以下のようになります。

```
数値を入力(1回目)12
数値を入力(2回目)20
0:加算 1:減算 2:乗算 3:除算
計算方法を入力 2
iNum1*iNum2=240
```

関数ポインタを使用

上記の四則演算を行うプログラムは、関数ポインタを使用して書き直すと以下のようになります。

コード

```
#include <stdio.h>

void func_add(int iNum1, int iNum2)
{
  /* 引数 iNum1 と iNum2 の加算結果を出力する */
  printf("iNum1+iNum2=%d\n", iNum1 + iNum2);
}
void func_sub(int iNum1, int iNum2)
{
  /* 引数 iNum1 と iNum2 の減算結果を出力する */
  printf("iNum1-iNum2=%d\n", iNum1 - iNum2);
}
void func_mul(int iNum1, int iNum2)
{
  /* 引数 iNum1 と iNum2 の乗算結果を出力する */
  printf("iNum1*iNum2=%d\n", iNum1 * iNum2);
}
void func_div(int iNum1, int iNum2)
{
  if (0 == iNum2) {
    /* 引数 iNum2 が 0 の場合はエラー */
    printf("Error.\n");
  }
  else {
    /* 引数 iNum1 と iNum2 の除算結果を出力する */
    printf("iNum1/iNum2=%d\n", iNum1 / iNum2);
  }
}

int main(void)
{
  int  iInput1;
  int  iInput2;
  int  iExpr;
  void  (*pFunc[])(int, int) = {func_add, func_sub, func_mul, func_div};

  printf("数値を入力(1回目)");
  scanf("%d", &iInput1);

  printf("数値を入力(2回目)");
  scanf("%d", &iInput2);

  printf("0:加算 1:減算 2:乗算 3:除算\n");
  printf("計算方法を入力 ");
  scanf("%d", &iExpr);

  switch (iExpr) {
  case 0:
  case 1:
  case 2:
  case 3:
    (*pFunc[iExpr])(iInput1, iInput2);
    break;
  default:
    /* 範囲外の入力が行われた */
    break;
  }
  return 0;
}
```

結果(例 1)

以下の入力を行なった場合の結果を示します。

1. 1 つ目の数値として 125 を入力
2. 2 つ目の数値として 92 を入力
3. 計算方法として 0 を入力(加算)

結果は以下のようになります。

```
数値を入力(1回目)125
数値を入力(2回目)92
0:加算 1:減算 2:乗算 3:除算
計算方法を入力 0
iNum1+iNum2=217
```

結果(例 2)

以下の入力を行なった場合の結果を示します。

1. 1 つ目の数値として 12 を入力
2. 2 つ目の数値として 20 を入力
3. 計算方法として 2 を入力(乗算)

結果は以下のようになります。

```
数値を入力(1回目)12
数値を入力(2回目)20
0:加算 1:減算 2:乗算 3:除算
計算方法を入力 2
iNum1*iNum2=240
```

ポインタ変数 pFunc は 配列になっており、宣言時に次のように設定されます。

- pFunc[0] に func_add 関数を登録
- pFunc[1] に func_sub 関数を登録
- pFunc[2] に func_mul 関数を登録
- pFunc[3] に func_div 関数を登録

関数ポインタの添え字 iExpr の値によって呼び出される関数が変化します。
関数の呼び出しは次のように 1 行で書くことができます。

```
(*pFunc[iExpr])(iInput1, iInput2);
```

まとめ

関数ポインタは、単独で使用するとあまりメリットはありませんが、例のように関数ポインタの配列として使用すると便利です。
配列として使用する場合は**登録する関数の復帰値、引数の型、引数の数を合わせる必要があります。**

目次

copyright (c) 日本システム開発株式会社 All Rights Reserved