ゲームプログラミング

▶　基礎編－第９回　ゲーム関数の呼び出し　◀

ゲームプログラムでは、場面ごとにそれを処理する関数を用意しておきます。ゲームがどこまで進んでいるかを判断し、これらの関数のうち適切なものを呼び出す必要があります。

場面にあわせた関数の呼び出し

ゲームプログラムでは、ゲームの場面にあわせた処理を行う必要があります。たとえば、オープニング、ゲームのメイン処理、エンディングなどの場面が考えられる場合、これらを処理するための関数を作成しておき、場面にあった関数を呼び出します。たとえば、タイトルを処理するTitle関数、メイン部分を処理するGameMain関数、エンディングを処理するEnding関数を作成しておき、ゲームがどこまで進んでいるかを判断してこれらを呼び出します。
ゲームがどこまで進んでいるかを判断し、場面にあった関数を呼び出すもっとも簡単な方法は、ゲームの状況を変数に格納しておき、これを参照してどの関数を呼び出すのかを決定する方法です。

```
// ゲーム処理関数
void GameFunc()
{
    // nGameModeはゲームの状況を格納する変数
    switch(nGameMode) {
      case 0:           // タイトル
        Title();        // タイトルを処理する関数の呼び出し
        break;

      case 1:           // ステージ１
        Stage1();       // ステージ１を処理する関数の呼び出し
        break;

      case 2:           // ステージ２
        Stage2();       // ステージ２を処理する関数の呼び出し
        break;

      case 9:           // エンディング
        Ending();       // エンディングを処理する関数の呼び出し
        break;
    };
}
```

この方法は、とても簡単なアルゴリズムであり、多くの初級ゲームプログラマが使用しています。しかし、ゲームの状況がたくさんある場合は、それにあわせてたくさんの分岐を記述することになります。ゲームの状況が１００ほどある場合は、case文やif～else文を１００分岐させることになります。さらに、GameFunc関数が呼び出されるたびに、どの関数を呼び出すかをいちいち判断する必要があるので、効率のよいものとはいえません。
ゲームの状況にあった関数を、効率的に呼び出すの方法の１つに、「関数のアドレスを使った関数の呼び出し」があります。

関数のアドレスと呼び出し

関数を呼び出すには、関数名にカッコをつけ、引数を指定することにより行います。たとえば、引数のないFunc関数を呼び出すには、

```
Func();
```

とします。コンピュータ内部では、Func関数を呼び出すという作業は「Func関数が配置されているアドレスを呼び出す」という処理になります。たとえば、Func関数が0x80000000番地に配置されている場合は、「0x8000000番地のプログラムを呼び出す」ことでFunc関数の呼び出しを実現しています。
すべての関数にはアドレスが割り当てられます。アドレスが判明している関数は、関数名がわからなくても呼び出すことができます。たとえば、0xF0000000番地に関数が配置されていることがわかっていれば、

```
0xF0000000();
```

とすることでその関数を呼び出すことができます(正確にはキャストが必要)。アドレスはポインタとして変数に格納することが可能なので、アドレスが格納できる変数(たとえば変数FuncPtr)に、関数のア

ドレスが格納されているとすれば、

```
FuncPtr();
```

と記述しても関数を呼び出すことができます。
　これを、「ゲームの状況にあわせた関数の呼び出し」に利用します。実行したい関数のアドレスを変数に入れておけば、

　　変数に格納されているアドレスの関数を呼び出す

という動作のみでゲームの状況にあった関数を呼び出すことができます。こうすれば、いちいちswitch文やif文でどの関数を呼び出すのかを判断する必要がなくなります。
　このような関数のアドレスを使った呼び出しは、「関数の切り替えが簡単に行える」「構造体に入れてデータと関数を一体化できる」という利点があり、多く利用されています(ちなみにC++では、仮想関数があるので、利用する場面はかなり減ります)。

関数のアドレスを格納する型の定義

　C言語には、関数のアドレスを格納するための型はないので、プログラマが定義する必要があります。関数の戻り値の型と引数の性質によって定義方法が異なります。たとえば、戻り値と引数のない関数のアドレスを格納するための"GameFuncPtr型"を定義するには、typedefを使って以下のようにします。

```
typedef void (*GameFuncPtr)();
```

　わかりにくい記述ですが、意味はGameFuncPtr型は、戻り値がvoid型で引数のない関数のアドレス(ポインタ)を格納する型である、となります。
　戻り値がdouble型、引数にint型の変数が２つの場合は、以下のように定義します。

```
typedef double (*GameFuncPtr)(int, int);
```

　このように定義したGameFuncPrt型の変数を宣言するには、通常の変数と同じように、

```
GameFuncPtr   GameFunc;
```

とします。これでGameFuncPrt型の変数GameFuncが宣言され、プログラムで使用できます。この変数には関数のアドレスを格納することができます。関数のアドレスは関数名から取得することができます。変数GameFuncへ関数のアドレスを代入するには、通常の変数と同じように=演算子を使います。たとえば、Title関数のアドレスを代入するには、

```
GameFunc = Title;
```

とします。ただし、型定義時に宣言した「戻り値と引数の性質」がまったく同じ関数のアドレスしか代入できません。
　変数GameFuncに格納されているアドレスから関数を呼び出すには、「関数のアドレスが格納された変数名」に「()」つけることで行います。

```
GameFunc();
```

　これは、通常の関数呼び出しまったく同じです。これで、変数GameFuncに格納されている関数を呼び出すことができます。変数GameFuncの内容を、ゲームの状況にあわせた関数のアドレスに書き換えることで、目的の関数を簡単に呼び出すことができます。

○ 課題 ○

　メッセージループを改良し、メッセージを処理したあとにゲームを処理する関数が呼び出されるようにプログラムを変更しましょう。

(１)GameFunc.hに、ゲームを処理する関数のアドレスを格納する独自の型(GameFuncPtr型)を定義します。GameFuncPtr型に代入できる関数のアドレスは、戻り値と引数がない関数だけになります。

```
/*****************************************************************************************************/
/*                                    ゲーム関数のアドレスを格納する型の定義                                    */
/*****************************************************************************************************/
typedef void (*GameFuncPtr)();
```

(２)GameFunc.cppの「外部公開変数」に関数アドレスを格納する変数GameFuncを宣言します。この変数は、WinMain.cppでも参照するので、staticを付けて宣言してはいけません。

```
GameFuncPtr    GameFunc = NULL; // 実行するゲーム関数のアドレスを格納
```

(３)WinMain.cppでは、GameFunc.cppで宣言した変数GameFuncを使ってゲーム処理関数を呼び出します。以下のようにexternをつけて変数を宣言することにより、GameFunc.cppで宣言した変数を参照できるようになります。

```
/*****************************************************************************************************/
/*                                        外部参照変数                                               */
/*****************************************************************************************************/
extern GameFuncPtr    GameFunc;        // 実行するゲーム関数のアドレスを格納
```

※externは、ほかのソースファイルで宣言されている外部変数を参照するときに使います。ただし、static宣言されている外部変数を参照することはできません。

(４)メッセージループ後の「ゲームを処理する関数の呼び出し」を、以下のように変更します。

```
        GameFunc();
```

変数GameFuncは、ゲームの状況にあわせて書き換えます。こうすると、状況にあった関数が呼び出されるようになります。

(５)GameFunc.cppに、機能テスト用のTest関数とTestProc関数を作成します。Test関数は、機能テスト処理を初期化する関数、TestProc関数は、機能テストのメインループ関数で、機能テストの終了条件を満たすまで何度も呼び出されるメインループの関数です。
　このように「関数のアドレス」を使ったゲームプログラムでは、１つの場面につき初期化用の関数と何度も実行されるメインループ関数の２つの関数が必要になります。

```
/*****************************************************************************************************/
/*                                          機能テスト                                               */
/*****************************************************************************************************/
void Test()
{
    // ここに、機能テストのための初期化処理を記述します

    GameFunc = TestProc;   // 初期化が終わったら、メインループ関数へ
}

/*****************************************************************************************************/
/*                                      機能テストメイン処理                                         */
/*****************************************************************************************************/
void TestProc()
{
    // 機能テスト初期化後に、この関数に制御が移ります。
    // この関数は何度も実行される、いわゆるメインループです。

    // ここに、機能テストのためのメイン処理を記述します


    Sleep(15);
}
```

(６)(２)で宣言した変数GameFuncはNULLが代入されており、このまま実行するとアドレスNULLにある関数が呼び出されて強制終了してしまいます。ここを「最初に実行する関数」に書き換えます。

```
GameFuncPtr    GameFunc = Test; // 実行するゲーム関数のアドレスを設定。Test関数から実行
```