ソフトゼミＢ　第７回 ファイル読み込み

■　はじめに

ここではファイル読み込みについて学びます。ファイル読み込みを用いることで、初期値の設定を簡単にいじることができます。応用次第ではマップの読み込みなども可能です。ここではファイル読み込みを利用して初期値の設定ができるようにします。では実際にやってみましょう。

■　ファイル読み込み

まず、前回までのソースを修正しましょう。

init に**太字**部分を追加して、網掛けで示した部分をコメントアウトしてください。

```
// DXライブラリの初期化に失敗すると即終了
if( DxLib_Init() == -1 ){ return -1; }
// 初期値の設定 失敗時、即終了
if( loadData( "data/data.txt") == -1 ){ return -1; }

/*
game.dPlayerX = 240;
game.dPlayerY = 240;
game.dPlayerLife = 100;
game.dPlayerVelocity = 4;

game.enemyTime = 50;
game.dEnemyLife = 5;
game.dEnemyVX = 0;
game.dEnemyVY = 1;

game.dPlayerShotLife = 1;
game.dPlayerShotVX = 0;
game.dPlayerShotVY = -15;
```

(次ページへ続く)

```
game.dEnemyShotLife = 1;
game.dEnemyShotVX = 0;
game.dEnemyShotVY = 15;
*/

// 初期化
player.x = game.dPlayerX;
player.y = game.dPlayerY;
```

それでは次にファイル読み込みの準備をします。zemi_b のフォルダの中に data という名前のフォルダを作ってください。そして、data フォルダの中に data.txt ファイルを作ってください。data.txt に以下のように書き込んでください。
※行頭の「；」は見やすくするために全角で記載していますが、実際には**半角**です。

```
；<<プレイヤー>>初期 x 座標  初期 y 座標  ライフ  速度
240 240 100 4
；<<敵>>発生間隔  ライフ  速度 x 成分  y 成分
50 5 0 1
；<<プレイヤー弾>>ライフ  速度 x 成分  y 成分(マイナス値)
1 0 -15
；<<敵弾>>ライフ  速度 x 成分  y 成分(プラス値)
1 0 15
```

最後に function.h に下記**太字**のプロトタイプ宣言を追加してください。

```
/* init.cpp */
int init ( void );
int loadData( char filePath[] );
```

これで準備は完了です。(ｹｶﾞｶｯﾀ･･･)

さあファイル読み込みを実装しましょう。以下のプログラムを init の 1 番下に書き込んでください。網掛け部分は印刷の都合上 2 行になっていますが、改行せずに 1 行で書いてください。

```
int loadData( char filePath[] ){
  int i = 0;
  int error = 0;
  int varNum;
  int file = FileRead_open( filePath );
  char line [ 256 ];
  while( FileRead_gets(line, 256, file) != -1 ){
    if( line[ 0 ] == ';' || line[ 0 ] == '¥0'){ continue;}
    switch( i ){
      case 0: // player
        varNum = sscanf_s( line, "%d%d%d%d", &game.dPlayerX, &game.dPlayerY,
&game.dPlayerLife, &game.dPlayerVelocity, 256 );
        if( varNum != 4 ){ error = 1;}
        break;
      case 1: // enemy
        varNum = sscanf_s( line, "%d%d%d%d%d", &game.enemyTime, &game.dEnemyLife,
&game.dEnemyVX, &game.dEnemyVY, 256 );
        if( varNum != 4 ){ error = 1;}
        break;
      case 2: //playerShot
        varNum = sscanf_s( line, "%d%d%d", &game.dPlayerShotLife, &game.dPlayerShotVX,
&game.dPlayerShotVY, 256 );
        if( varNum != 3 ){ error = 1;}
        break;
      case 3: //enemyShot
        varNum = sscanf_s( line, "%d%d%d", &game.dEnemyShotLife,
&game.dEnemyShotVX, &game.dEnemyShotVY, 256 );
        if( varNum != 3 ){ error = 1;}
        break;
    }
    if( error ){ break;}
    (次ページへ続く)
```

```
    i++;
  }
  FileRead_close( file );
  if( error ){ return -1; }
  return 0;
}
```

ここからプログラムの大雑把な解説をしていきます。(詳しい関数解説は後の方で。)
FileRead_gets はファイルを開くための DX ライブラリ関数です。これでさっきの date.txt から**文字列**を**行単位で**読み込むことができます。sscanf_s は文字列を読み込むための関数です。この場合 1 列の文字を読み込んで game.dPlayerX などに当てはめています。FileRead_close で開いていたファイルをとじます。ちゃんと閉じないとエラーが出るので忘れないようにしましょう。行頭の「;」の後の 1 列は読み込まないようになっているので、注意してください。(コメントアウトみたいなもん)

■ 文字列

ゼミ A で扱いきれなかったのでここで説明。

C 言語(と C++)で、「文字列」とは「文字の配列」で、char 型の変数の配列で表されます。半角文字は char 型変数を 1 つ使って表します。(全角文字についてはゼミ B の領域を若干超えるのでここでの説明は避けます。)

```
char str[256];
```

という宣言で、半角文字が(256-1=)255 文字分入る配列が宣言されます。文字列の最後の 1 要素分(str[255])は NULL 文字と呼ばれる「**文字列の終わりを表す符号**」(数字の 0)が入ります。)

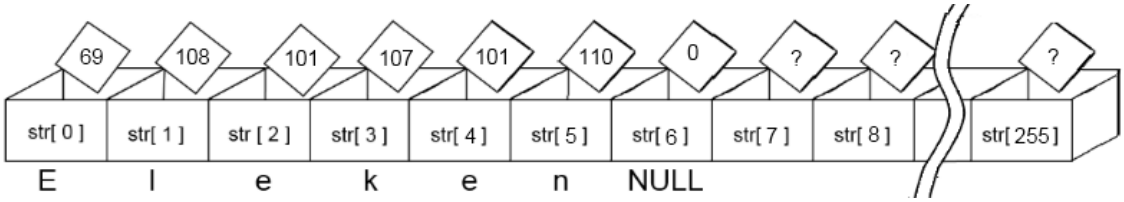
半角文字は ASCII(アスキー)と呼ばれる規則で、文字ごとにある決まった数字が割り当てられています。たとえば半角英大文字の「A」は「65」(=16 進数で 41)、半角英小文字の「e」は「101」(=16 進数で 65)という値を持ちます。配列 hoge の**初期化時**に(初期化時以外では代入できないので注意!)

```
char str[256] = “Eleken”;
```

と宣言すると、str[0]には大文字の'E'を示す 69(=16 進数で 45)
str[1]には小文字の'l'を示す 108(=16 進数で 6C)
…
str[5]には小文字の'n'を示す 110(=16 進数で 6E)
str[6]には文字列の終端を示す 0(=16 進数で 0, NULL 文字)
がそれぞれ代入されます。(str[7]～str[255]はゴミが入っていると考えてください。そし

て、NULL 文字以降のゴミたちは文字列操作においては全て無視されます。)

ゼミ A で配列を説明する時に用いた図を使って表すと、上の初期化後の状態は次のように示されます。(次ページ)



繰り返しますが、この代入は「**配列の初期化時のみ有効**」です。

その他の場面で、例えば、

```
str = “mofmof”;
```

のような代入は**絶対に絶対に絶対に絶対にできない**ので注意してください。※C++の「string」という機能を使うとできますが、ややこしいのでここでは取り扱いません。詳しくはゼミ C で取り扱う予定です。

文字列のコピーを行いたい場合は strcpy_s(要 #include<string.h>(もしくは #include<cstring>))や sprintf_s(要#include<stdio.h>(もしくは#include<cstdio>))などの関数が必要です。(使い方は聞いてください。) (こちらも、C++の string を使うと簡単に書けますが、ゼミ C で扱います。)

なお、終端を示す符号である NULL 文字ですが、文字の「'0'」(10 進数で 48, 16 進数では 30 で示される)とは全くの別物です。注意してください。

また、ソースコード中で文字を一重引用符(シングルクォート, ‘←これ)でくくると、それに相当する数字と同じ効果を持ちます。例えば、'A' ←これは(ASCII または ASCII をもとにした文字コードの場合、) 「65」(=16 進数で 41)と同じ意味を持ちます。

ASCII コードについては、今回配布した標準ライブラリ関数のプリントの後ろの方に一覧を掲載しています。

説明が長々となってしまいましたが、上のサンプルコード(init.cpp)には、
int loadData(char filePath[])という関数があります。
これは引数として文字列(char 型配列)を受け取っています。実際に、main.cpp から

```
if( loadData( "data/data.txt") == -1 ){ return -1; }
```

というふうに文字列を引数として渡しています。

そして、init.cpp の while ループ中にある

```
if( line[ 0 ] == ';' || line[ 0 ] == '¥0'){ continue; }
```

では、**その行の先頭が「;」または NULL 文字の時はその行は読まない**、ということをやっています。その行の文字列が NULL 文字だけで表されるということは、その行には「改行」

しかなかったために、FileRead_gets 関数が何もないことを示すために NULL 文字を line[0]に入れたというわけです。(ちなみに、改行は「¥r¥n」(Windows)もしくは「¥n」(Mac OS X, Linux)もしくは「¥r」(古い Mac)という特殊文字で表されるのですが、今回は扱いませんでした。)

■ 新出関数

➢ ファイル入出力に関する関数

FileRead_open 関数

int FileRead_open(char *FilePath) ;

ファイルを開きます。**FilePath** はファイルへのパスを示す文字列です。ゼミ B では、文字列を「char 型の配列」と解説しましたが、ゼミ∀で示した通り、「配列はポインタ」なので、char FilePath[]と char *FilePath はだいたい同じです。

FileRead_gets 関数

int FileRead_gets(char *Buffer, int Num, int FileHandle) ;

開いたファイルの 1 行を読みます。繰り返し呼ぶことで、そのファイルを読み進めることができます (詳しくは後述) 。第 1 引数の Buffer には、読み取った文字列を格納する char 型配列(もしくはポインタ)を、第 2 引数の Num にはその char 型配列の大きさを、第 3 引数の FileHandle には、開いたファイルの識別番号(ファイルハンドル, FileRead_open の戻り値)を入れてください。普段は読み取った文字列の長さを戻り値として返しますが、ファイルの終端に達した場合には「**-1**」という特別な値を返します。

FileRead_close 関数

int FileRead_close(int FileHandle) ;

FileRead_open で開けたファイルを閉じます。「開けたら閉める」を忘れずに。

具体例

例えば、以下のようなテキストファイル「example.txt」があったとします。

```
HogeFugaFooBar
Mofmofmofmof


AAAAAAAAAA
```

また、読み取った文字列(行)を格納する十分大きい配列も用意しておきます。

```
char buf[ 256 ];
```

まず、open してファイルハンドルを適当な int 型変数に入れます。

```
int file = FileRead_open( “example.txt” );
```

試しにファイルから 1 行読んでみます。

```
FileRead_gets( buf, 256, file );
```

すると、配列 buf の中身は” HogeFugaFooBar”となっています。(もちろん最後に NULL

文字がいることを忘れずに。）もう一度

```
FileRead_gets( buf, 256, file );
```

すると、buf には「Mofmofmofmof」が入ります。さらに同じステップで次の行も読み込んだとします。次の行は何も書かれていない「空行」です。buf には buf[0]に NULL 文字が入り、空の文字列が表現されます。さらにもう 1 行読み込むと、buf は「AAAAAAAAAA」となります。これで、ファイルの末尾に達しました。これ以上読み込もうとしても読み込めないので、これ以降は戻り値として「-1」を返します。ですから、次のような処理をすると、ファイルの中身を全て読むことができます。

```
char buf[ 256 ];
int file = FileRead_open( “example.txt” );
while(FileRead_gets( buf, 256, file ) != -1 ){
  /*読み取った buf に何らかの処理をする*/
}
FileRead_close( file );
```

- 文字列操作に関する関数

sscanf_s 関数 ※stdio.h(もしくは cstdio)

int __cdecl sscanf_s(const char* _Src, const char* _Format);

第 1 引数 Src には、文字列(char 型配列)を引数として渡します。第 2 引数では、いつもの scanf 同様に、フォーマットの指定(%d など)を行い、第 3 引数以降では、読み取る変数のアドレスを渡します。そして、最後の引数として、第 1 引数に指定した char 型配列の大きさを指定する。すると、なんと文字列から scanf ができてしまうのです。

例えば、

```
int a, b, c;
char hoge[ 100 ] = “30 20 10”;
```

のように、3 つの int 型変数が宣言されていて、数字が半角スペース区切りで文字列として与えられていたとします。これを関数無しで hoge から 30,20,10 を読み取って a,b,c に代入するにはかなり手間がかかります。そこで、

```
sscanf_s( hoge, “%d%d%d”, &a, &b, &c, 100 );
```

とすると a に 30 が、b に 20 が、c に 10 が代入されます。(最後の 100 は配列 hoge の大きさ) sscan_f、それは文字列に対する scanf なのです。

なお、使うには stdio.h(もしくは cstdio)のインクルードが必要ですが、DxLib.h が既に stdio.h をインクルードしているので、DX ライブラリを使っている場合には改めてインクルードする必要はありません。

戻り値は、読み取った変数の個数です。整数を要求しているのに整数値以外が出てきた

り、数が足りなかったりしたら数が変化します。

■ 終わりに

これで、ゼミ B の前半部分のシューティングは終了です。時間が余っているならば、以下のような研究をしてみると、実際の制作の参考になるかもしれません。

- タイトル画面の追加
- 弾を自機狙いに改造する
- 弾が円軌道や、正弦関数の軌道をとるように動くように改造する
- それぞれの弾にランダムの速度をつけてみる
- ボスが出現するようにする
- ゲームクリアの処理をつくる
- 背景画像を追加する
- 効果音をつける
- スコアランキングをつくる

■ 次回は…

本来は、次回から 3 回にかけてアクションゲームを制作する予定でしたが、進度調整を行ったり、台風が直撃したりした結果、縮小せざるを得ない状態となってしまいました。というわけで、次回はアクションゲームの最も重要な部分である「重力」とプラスアルファの要素を少しだけ扱うこととします。

また、アクションゲームに関しては時間があれば夏休み中にゼミ C で取り扱います。今回のゼミ B はいろいろな要素を詰め込みすぎたため、資料が複雑怪奇なものになってしまったので、ゼミ C ではそのようなことにならないよう努力していきます。