BREW & EZアプリ

ゲームプログラミング

○　第６回　文字の描画　○

文字配列

C言語での文字列は、ダブルクォーテーション「"」で囲まれた部分を文字列として扱います。この文字列そのものを、C言語では変数に代入することができません。C言語には文字列型という型が存在しないからです。しかし、これでは大変不便です。そこで、char型と配列を利用します。

"Tadashi"という文字列を文字配列に格納することを考えてみましょう。この文字列は、'T', 'a', 'd', 'a', 's', 'h', 'i'で７文字あります。しかし、C言語での文字列は、最後を意味する終了文字として、NULL文字('\0'と表します)を入れることになっています。したがって、"Tadashi"を格納するのに必要な文字数は合計８文字となります。これを格納するための配列は、以下のように定義します。

```
char    name[8];
```

この配列をchar型の配列、または文字配列と呼びます。文字配列には、文字列が何文字で構成されているかという情報はいっさい記憶しません。NULL文字が現れた時点で文字列の終了とみなしているだけです。

文字列の初期化

以下のように、文字配列に"Tadashi"を格納することを考えてみましょう。

T	a	d	a	s	h	i	\0
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

char型の配列を文字で初期化するので、以下のように記述することができます。

```
char    name1[8] = {'T', 'a', 'd', 'a', 's', 'h', 'i', '\0'};
```

しかし、これでは面倒なので、文字列に限って以下のような形式で初期化することができます。

```
char    name2[8] = "Tadashi";
```

また、配列の要素の数を指定しないで、初期化で指定された数だけメモリを確保することもできます。

```
char    name3[] = {'T', 'a', 'd', 'a', 's', 'h', 'i', '\0'};
char    name4[] = "Tadashi";
```

上記の４つの配列は、結果的にまったく同じデータに初期化されます。しかし、以下の２つの場合は先頭の４文字は「Ando」となり、残りの４文字はNULL文字に初期化されます。

```
char    name5[8] = {'A', 'n', 'd', 'o'};
char    name6[8] = "Ando";
```

ただし、配列の大きさを省略して、

```
char    name7[] = {'A', 'n', 'd', 'o'};
```

とすると、この配列は４文字分の大きさを持ち、先頭から'A', 'n', 'd', 'o'の文字で初期化されますが、NULL文字はないことに注意してください。NULL文字がないと、文字列の終わりがわからなくなってしまいます。試しに、上記のname7を表示してみると、５文字目以降に不定の文字が表示されるはずです。NULL文字が現れるまで、配列の先頭からメモリの内容を表示し続けてしまっているのです。

では、以下の初期化はどうなるでしょう。

```
char    name8[7] = {'T', 'a', 'd', 'a', 's', 'h', 'i', '\0'};
char    name9[7] = "Tadashi";
```

name8には、７文字分しか割り当ててないのもかかわらず、８文字の初期化を与えているのでコンパイルエラーになります。しかし、name9の初期化はコンパイルエラーになりません。文字列で初期化す

る場合は、「文字列にNULL文字が出現する」または「文字配列を埋め尽くす」のどちらかを満たすことで終了するという規則があるのです。この場合、name9には後者の規則が適用されるので、文字列"Tadashi"で配列が埋められてしまいます。その結果、文字列の終了を示すNULl文字のない文字配列ができてしまいます。

なお、文字列が代入できるのは初期化のときだけです。次のようなプログラムは文法上エラーになってしまいます。

```
char    name[256];

name = {'T', 'a', 'd', 'a', 's', 'h', 'i', ' ', 'A', 'n', 'd', 'o', '\0'};
name = "Tadashi Ando";
```

文字列の操作

C言語の文字列は、char型の配列で表現するため、配列の規則が適用されます。そのため、=演算子で文字列を代入したり、+演算子で文字列を連結したり、不等号演算子で文字列の比較を行うことはできません。たとえば、以下のようなプログラムは、エラーになったり、正しく動作しません。

```
char    str1[] = "Ando";
char    str2[] = "Tadashi";

char    str3[] = str1 + str2;     // エラー

if(str3 > str2)                   // 正しく動作しない
    printf("文字列str3の方が大きい");
```

文字列をコピーするのに、各要素ごとに文字をいちいち代入していては、かなり面倒です。この問題を解決するため、C言語には、文字列を扱うための標準関数がたくさん用意されています。しかしBREWではC言語の標準ライブラリ(BREW以外のライブラリ)を使用してはならない、という規則があります。その代わりに、C言語の標準関数と同等のことを行うヘルパー関数が用意されています。文字列操作の代表的なヘルパー関数を以下に紹介します。

STRLEN	文字列の長さ(文字数)を返します。終端のNULL文字は含まれません
STRCPY	文字列のコピーを行います
STRNCPY	文字列のコピーを行います。コピーする最大文字数を指定できます
STRCMP	文字列の比較を行います
STRNCMP	文字列の比較を行います。比較する最大文字数を指定できます
STRCAT	文字列の連結を行います
STRNCAT	文字列の連結を行います。連結する最大文字数を指定できます
STRCHR	文字列から文字を検索します
STREXPAND	文字列をワイド文字列に変換します
SPRINTF	printfの出力先が文字配列になったものです
DBGPRINTF	デバッグ出力を行います

これらのヘルパー関数は、ヘッダファイル"AEEStdLib.h"をインクルードすることにより使用できるようになります。また、いくつかの関数では注意が必要です。C言語の標準関数と同様に、配列の境界チェックを行っていません。たとえば、STRCPY関数やSTRCAT関数です。これらの操作により配列から文字があふれてしまった場合は、メモリを破壊してでも処理が続けられ、なおかつエラーも返しません。文字配列の要素数をあらかじめ多めに確保しておくことである程度回避できますが、完全ではありません。

ワイド文字列

BREWにはAECHAR型という、文字を格納するもうひとつの型があります。char型の配列は、半角１バイト、全角２バイトで格納するマルチバイト文字列と呼ばれるものです。これに対してAECHAR型の配列では、すべての文字を２バイトで格納します。これをワイド文字列と呼びます(同じワイド文字を格納するUnicode(wchar_t型)がありますが、AECHAR型とは全く異なるもので、BREWでは使われていません)。

AECHAR型の文字列は、文字を画面に描画するIDISPLAY_DrawText関数で必要になります。AECHAR型の配列は、宣言時の代入で初期化できないので、次のようにSTREXPAND関数を用いてchar型文字列をワイド文字列に変換しなければなりません。

```
char*    text = "御迦茂斗 夜死鬼";
AECHAR   wtext[128];

// char文字列からワイド文字列へ変換
STREXPAND(text, STRLEN(text), wtext, sizeof(wtext));

// 文字列描画
IDISPLAY_DrawText(pMe->pIDisplay,
                  AEE_FONT_NORMAL,                          // フォント
                  wtext,                                    // 描画文字列(ワイド文字列)
                  -1,                                       // 描画文字数(-1はすべて)
                  0,                                        // 描画ｘ座標
                  0,                                        // 描画ｙ座標
                  NULL,                                     // 描画領域
                  IDF_ALIGN_CENTER | IDF_ALIGN_MIDDLE); // 描画フラグ
```

前述のヘルパー関数は、ワイド文字列には使用できません。代わりに、以下のようなワイド文字列用のものを使用します。

WSTRLEN	STRLEN関数のワイド文字列版。ただし、全角、半角ともに１文字として数えます
WSTRSIZE	ワイド文字列(の先頭からNULL文字まで)のサイズ(バイト数)を返します
WSTRCPY	STRCPY関数のワイド文字列版。より安全なWSTRLCPY関数もあります
WSTRNCOPYN	STRNCPY関数のワイド文字列版。WSTRNCPYではないことに注意
WSTRCMP	STRCMP関数のワイド文字列版
WSTRNCMP	STRNCMP関数のワイド文字列版
WSTRCAT	STRCAT関数のワイド文字列版。より安全なWSTRLCAT関数もあります
WSTRCHR	STRCHR関数のワイド文字列版
WSPRINTF	SPRINTF関数のワイド文字列版
WSTRCOMPRESS	STREXPAND関数とは逆に、ワイド文字列をchar文字列に変換します

練習問題

1　char型の変数cを宣言し、そこに'A'という文字を代入するプログラムを作成しましょう。

2　１２８文字格納できる文字配列strを宣言しましょう。

3　文字配列strを"御迦茂斗 夜死鬼"という文字列で初期化するプログラムを作成しましょう。

4　文字配列strに格納されている文字列が何文字あるか数えるプログラムを作成しましょう。

5　１２８文字格納できる文字配列str2を宣言しましょう。

6　文字配列str2のすべての文字をNULL文字にするプログラムを作成しましょう。

7　文字配列str1の文字列を文字配列str2にコピーするプログラムを作成しましょう。

8　２５６文字格納できる文字配列str3を宣言しましょう。

9　文字配列str1の文字列の後ろに文字配列str2の文字列を連結した文字列を、文字配列str3に格納するプログラムを作成しましょう。

課　題

画面に文字を描画しましょう。

(１)ヘルパー関数を扱うためのヘッダファイルをインクルードしましょう。

(２)*ププロジェクト名*_Draw関数に、文字列「御迦茂斗 夜死鬼」を座標(0，0)に描画するプログラムを作成しましょう。

(３)描画座標を(10，20)に変更しましょう。

(４)画面中央に描画されるようにプログラムを変更しましょう。

(５)フォントをラージフォントに変更しましょう。

(６)文字の色を青(RGBA[0，0，255，255])に変更しましょう。