char 型を攻略してみる

Borland C++ Builder では文字列型を使いますが、C++ Builder 以外の環境では C 言語からの char 型をよく使います。

文字型データ

データ型	呼称	ビット幅	範囲例
char	文字型	8	－128 ~ 127
signed char	符号あり文字型	8	－128 ~ 127
unsigned char	符号なし文字型	8	0 ~ 255

char 型は文字型と呼ばれますが、これは ASCII 文字セットを表現するのに十分かつ無駄のないサイズを持っているからです。特別に文字を扱うための便利な機能を持っているわけではありません。char 型は、

―――文字表現に都合のよいビット幅を持った整数型です。

char 型の「符号有無」は実装依存なので、符号のありなしで動作の変わってしまう記述には注意が必要です。

例） getchar()の返り値はエラーの場合－1 なので符号なしの unsigned char に代入出来ません。

また、ASCII 文字セットは英数字、特殊文字および制御文字からなり、漢字に関する規定がありません。

- 宣言

  char 型の宣言の仕方は 3 つあります。

```
① char txt;
② char txt[文字数];
③ char txt[] = “Hello World!”;
```

① この場合の宣言では 1 文字分の文字型を宣言したことになります。

② この場合の宣言では文字列の値－1 分の文字列を入力することが出来ます

この－1 とは char 型で文字列を表わす場合、文字列の終わりを示すために最後の配列に'¥0'（NULL 文字）が入ります。

③ 宣言時に文字列を代入することで、自動で配列を確保してくれます。

この場合 13 文字分の配列が用意されます。

Hello □ W o r l d ! ¥0

例）char txt[] = “Hello World!”;

txt[0]には「H」が

txt[1]には「e」が入っています。

- 入力

String 型では文字を代入する場合

```
String txt;
txt = “Hello World!”;
```

これで代入することが可能です。

しかし、char 型の場合

```
char txt[64];
txt = “Hello World!”;
```

これだとエラーが出てしまいます。

また

```
char txt[64],str[] = “Hello World!”;
txt = str;
```

の場合でもエラーが出てしまいます。

String 型に慣れている人が最初に躓くのはここだと思います。

これを解決するには C 言語の標準ライブラリ関数を使います。
この場合は strcpy を使います。

- 標準ライブラリ関数
  紹介一覧
  strcpy　：文字列のコピー
  strcat　：文字列の連結
  strlen　：文字列の長さの取得
  strstr　：文字列１から文字列２を検索する
  sprintf　：書式指定変換した出力を文字列に格納

  - strcpy
    【必須ヘッダー】
    <string.h>
    【書式】
    char *strcpy(char *s1, const char *s2);
    【説明】
    文字列のコピー
    文字型配列 *s1 に文字列 *s2 を '¥0' までコピーします。'¥0' もコピーするので s1 はその分も考えて大きさを宣言しておかなければなりません。
    もし、s1 と s2 が重なっている場合には動作は未定義となります。
    【引数】
    char *s1　：　複写先の文字型配列
    const char *s2　：　複写する文字列
    【戻り値】
    s1 の値。つまり戻り値はコピー後の文字列を指す。
    【使用例】

```
char str1[64] = “Hello”,str2[64] = “World”;
strcpy(str1,str2);
```

これにより「str1」には「World」が入っています。

- strcat

【必須ヘッダー】

<string.h>

【書式】

char *strcat(char *s1, const char *s2);

【説明】

文字列の連結

文字型配列 s1 のうしろに文字列 s2 を連結します。'¥0' も連結するので s1 はその分も考えて大きさを宣言しておかなければなりません。

s1 を超して連結した場合と、s1 と s2 が重なっていた場合は、動作は未定義となります。

【引数】

char *s1　：　連結先の文字型配列

const char *s2　：　連結する文字列

【戻り値】

s1 の値。つまり戻り値はコピー後の文字列を指す。

【使用例】

```
char str1[64] = “Hello”,str2[64] = “World”;
strcat(str1,str2);
```

これにより「str1」には「HelloWorld」が入っています。

- strlen

【必須ヘッダー】

<string.h>

【書式】

size_t strlen(const char *s);

【説明】

文字列の長さの取得

文字列 s の長さを取得し返却します。長さに'¥0'は含みません。

【引数】

const char *s　：　文字列

【戻り値】

文字列 s の長さ。

【使用例】

```
char str1[] = “Hello World!”;
int n = strlen(str1);
```

これにより「n」には「12」が入っています。

- strstr

【必須ヘッダー】

<string.h>

【書式】

char *strstr(const char *s1, const char *s2);

【説明】

文字列１から文字列２を検索する

文字列 s1 の先頭から文字列 s2 を探し、見つかったときにはその位置をポインタで返却し、見つからなかったときには NULL を返却します。

【引数】

const char *s1　：　検索対象文字列

const char *s2　：　検索文字列【戻り値】

【使用例】

```
char str1[] = “Hello World!”,str2[] = “Wor”,*p;
p = strstr(str1,str2);
```

これにより「p」には「W」のポインタが入っています。

- sprintf

【必須ヘッダー】

<stdio.h>

【書式】

int sprintf(char *str, const char *format, ...);

【説明】

書式指定変換した出力を文字列に格納

書式 format にしたがって、printf 関数と同様の変換を行った出力を、文字列 str に格納します。

- 【引数】

char *str : 変換した出力を格納する文字列。

const char *format : 書式指定文字列。

... : 可変個引数。この引数を書式指定に従い変換します。書式指定文字列はこの引数と同数必要です。

【戻り値】

成功時 : str に格納した文字数（最後の'¥0'は除く）

失敗時 : EOF

【使用例】

```
char str[64] ,str1[] = “Remilia”;
int num = 500;
sprintf(str,” %s is about %d years old.”,str1,num);
```

これにより「str」には「Remilia is about 500 years old.」が入っています。

紹介した関数以外にも文字列に関する関数はたくさんあります。
気になった人は調べてみましょう。

strstr：文字列の検索

strcomp：文字列の比較

atoi：文字列を整数値に変換

itoa；整数値を文字列に変換