5.7 メモリの動的割り当て

データを格納するためのメモリの領域は通常は変数宣言によって確保される。外部変数や静的変数の領域はプログラムの実行を通して、自動変数の場合は必要が生じる度に割り当てられる。

一方、プログラムの途中で空いているメモリを割り当てる関数 (ヘッダファイルは stdlib) が用意されている。メモリの割り当てに成功すると割り当てられた領域の先頭番地を返す。割り当てに失敗したときは空ポインタ (null pointer) を返す。空ポインタは何も指してないポインタである。

一般形式	機能
`void *calloc(size_t nmemb, size_t size)`	各要素が size バイトを必要とする nmemb 個の要素からなる配列にメモリの連続領域を割り当てる。領域はすべてのビットが 0 になるように初期化される。
`void *malloc(size_t size)`	大きさが size バイトのメモリの領域を割り当てる。初期化は行わない。
`void free(void *ptr)`	ptr が指す領域を解放し、その後の割り当てに利用できるようにする。返却値はない。

```
/* pointer-string-function.c */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(void)
{
  char *str;

  str=malloc(100*sizeof(char)); /* str の指す文字列の領域を確保 */
  strcpy(str,"abc");
  printf("文字列%s の長さは%zu です。\n",str,strlen(str));

  free(str); /* 使い終わった領域を解放 */

  return 0;
}
```

実行結果は以下。

```
~/comp3a$ cc pointer-string-function.c -o pointer-string-function -Wall
~/comp3a$ ./pointer-string-function
文字列 abc の長さは 3 です。
```

5.8 文字列操作関数 — その 2

ライブラリの文字列操作関数 (string.h にある) と同等の関数を作る。本節の関数および解説は、

- Al Kelley and Ira Pohl, A Book on C:Programming in C, 4th ed., Addison Wesley, 1998 (若林訳、基本から学ぶ C 言語、日経 BP)
- 平林雅英著、ANSI C/C++辞典、共立

を参考にした。

5.8.1 strcpy()

```
char *mystrcpy(char *str1, const char *str2)
{
  char *p=str1; /* ポインタ p はポインタ値 str1 に初期化 */

  while ((*p++=*str2++)) /* == と区別するため代入式を括弧で囲った */
    ;

  return str1;
}
```

1. `char *p=str1`
   ポインタ p はポインタ str1 の保持するアドレスで初期化されている。したがって、p と str1 はメモリの同じ位置を参照している。
2. `*p++`
   後置演算子は間接参照演算子よりも優先順位が高いので`*(p++)` と同じで、p の現在の値が参照され、ついで p がインクルメントされる。
3. `*p++=*str2++`
   p が参照している文字に str2 に参照している文字が代入される。次いで、p と str2 がインクルメントされる。
4. `while ((*p++=*str2++))`
   str2 がナル文字'\0' を参照しない限り while 文は続けられる。p の参照先に 0 が代入された時点で while 文は終了する。

5.8.2 strcat()

```
char *mystrcat(char *str1, const char *str2)
{
  char *p=str1;

  while (*p!='\0') /* *str1 の最後のナル文字に到達 */
    ++p;

  while((*p++=*str2++)) /* *str2 を*str1 のナル文字以降にコピー */
    ;

  return str1;
}
```

1. `while(*p!='0')`
   `  ++p;`
   `while(*p!='\0')` は `while(*p)` とも書ける。p が参照している値が非零の場合に p をインクリメントし、文字列内の次の文字が参照される。p がナル文字'\0' を参照すると式*p の値が零になるので while ループを抜ける。

2. `while((*p++=*str2++));`
   この while 文の開始時に p は str1 が参照している末尾のナル文字を参照している。したがって、str2 内の文字をメモリに次々コピーしていけば、str1 の末尾のナル文字とそれ以降の内容が上書きされる。

5.8.3 strcmp()

```
char *mystrcmp(const char *str1, const char *str2)
{
  int i;

  while(*str1==*str2){
    if (*str1=='\0') /* 同じ長さですべて一致 */
      return 0;
    str1++;
    str2++;
  }

  i=(unsigned char)*str1-(unsigned char)*str2;
  return i;
}
```

5.8.4 strlen()

```
size_t mystrlen(const char *str)
{
  size_t i=0;

  while(*str1++!='\0')
    i++;

  return i;
}
```

`while(*str1++!='\0')` は `while(*str1++)` とも書ける。

5.8.5 文字列操作関数の利用例

```
/* mystring-function.c */
#include <stdio.h>
#include <string.h>

char *mystrcpy(char *str1, const char *str2);
char *mystrcat(char *str1, const char *str2);
int mystrcmp(const char *str1, const char *str2);
size_t mystrlen(const char *str);

int main(void)
{
  char *str;

  str=(char *) malloc(100*sizeof(char)); /* str が指す文字列の領域を確保 */

  strcpy(str,"abc");
  mystrcat(str,"defg");

  printf("str=%s\n",str);
  printf("%\n",mystrlen(str));

  printf("%d\n",mystrcmp(str,"abd"));
  printf("%d\n",mystrcmp(str,"abb"));

  free(str);

  return 0;
}
```

コンパイルと実行。

```
~/comp3a$ cc -o mystr mystring-function.c mystrcpy.c mystrcat.c（続く）
mystrcmp.c mystrlen.c
~/comp3a$ ./mystr
str=abcdefg
7
-1
1
```

ソースファイルが複数に分かれているときは、cc コマンドの引数にソースファイルを空白で分けて並べる。たとえば、5 つに分れているときコンパイルは次のように行う。オプションは cc とファイル 1 の間に置き前後に空白を入れる。

```
cc ファイル 1 ファイル 2 ファイル 3 ファイル 4 ファイル 5 -o 実行ファイル名
```

演習 5.7 max-array.c(p.30) をポインタを用いて書き直せ。

演習 5.8 例 3.6(p.32) をポインタを用いて書き直せ。

演習 5.9 演習 5.5(p.62) をポインタを用いて書き直せ。

演習 5.10 演習 5.6(p.62) をポインタを用いて書き直せ。