5　文字列、配列、ポインタ

5.1　文字列とポインタ

3.2 節 (p.37) では、文字列を char 型の配列として扱ったが、char 型へのポインタとして文字列を扱うこともできる。

次のプログラムは char 型へのポインタを文字列リテラルで初期化している。

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

int main(void)
{
  char *strPtr="abc";

  printf("文字列は%s です。\n", strPtr);

  return 0;
}
```

char *str で str を char 型へのポインタとして宣言し、メモリ上のどこかに設定された文字列"abc" の先頭アドレスでポインタ strPtr を初期化している。

演習 5.1 ”Hello world\n”を出力するプログラムをポインタを用いてつくれ。

char 型配列は文字列リテラルにより初期化することができるが、文字列を代入することはできない。一方、char 型へのポインタには文字列の代入ができる (実際には文字列の先頭アドレスをポインタ型変数に代入することである)。

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

int main(void)
{
  char *strPtr = "abc";

  printf("文字列は%s です。\n",strPtr);
  str="def";
  printf("文字列は%s です。\n",str);

  return 0;
}
```

5.2　型修飾子 const

型修飾子 const をオブジェクト宣言で用いるとオブジェクトを初期化できるが、後でその値を変更できなくなる。const 修飾子を積極的に用いることによって，定数を不用意に変更してしまうバグをさけることができる。

```
/* const.c */
#include <stdio.h>

int main(void)
{
  const int a=1;

  a++;
  printf("a の値は%d です。\n",a);

  return 0;
}
```

東女のコンパイラは”increment が読み込み専用オブジェクト ‘a’ に行われました” という意味の error メッセージを出し、コンパイルできない。

```
~/comp3a $ cc -Wall const.c -o const -Wall
const.c: In function 'main':
const.c:8: error: increment of read-only variable 'a'
~/comp3a $ ./const
-bash: ./const: No such file or directory
```

ポインタと const 修飾の関係は以下のようになる。

- char *p は char 型へのポインタである。型分類はポインタ型であって、char 型ではない。
- char * const p は char 型への読み込み専用のポインタである。(const 修飾された pointer 型)
- const char *p は const char 型 (const 修飾された char 型) へのポインタである。
- char const *p は const char *p と同様 const char 型 (const 修飾された char 型) へのポインタである。

5.3 文字列操作関数

C には文字列操作の演算子が存在しないので、文字列操作にはポインタを使うか文字列操作関数を使う。

string.h で定義された文字列を操作する関数の主なものには以下に示す。

const char * はメモリ内で参照している文字を変更してはならない char 型 (const 修飾された char 型) へのポインタでる。(5.2 節 p.57 参照) 文字列リテラルや変更してはならない char 型配列 (const 修飾された char 型の配列) の配列名がこれに該当する。size_t は符号なし整数型で大きさはシステムにより異なる[6]。

[6]大きさは sizeof(size_t) で与えられる。単位はバイト。

一般形式 (プロトタイプ)	機能
`char * strcpy(char *str1, const char *str2)`	文字列 str2 を str1 の領域に複写する。複写後の文字列を返す。
`char * strcat(char *str1, const char *str2)`	文字列 str2(文字列 2 の終端を示すナル文字を含む) の複写を文字列 str1 の最後に付加する。連結後の文字列を返す。
`int strcmp(const char *str1, const char *str2)`	文字列 str1 が文字列 str2 より大きいか、等しいか、小さいかによって、それぞれ 0 より大きい、0、0 より小さい整数を返す。最初の異なる文字を unsigned char 型として比較する。
`size_t strlen(const char *str)`	文字列 str の終端を示すナル文字に先行する文字の個数を返す。

東女の C コンパイラ cc は C99 仕様なので、`size_t` 型の変換指定子は`%zd` または`%zu` とする。

例 5.1 `strcpy()` を用いて、文字列を配列にコピーする。

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

int main(void)
{
  char str[100];

  strcpy(str,"abc");
  puts(str);

  return 0;
}
```

演習 5.2 キーボードから文字列を入力し、入力した文字列の長さを表示するプログラムを `strlen()` 関数を使って作れ。たとえば、”This is a string.” と入力すると”17 文字” と出力する。(改行コードは含めない。) 文字列操作関数の使い方は、上の `string-assignment.c` を参考にせよ。

演習 5.3 キーボードから英文を入力し、スペース、カンマ、ピリオドの数を数えるプログラムをつくれ。スペース、カンマ、ピリオドのアスキーコードは、16 進で 0x20,0x2c,0x2e である。(Hints. 入力した文字列の長さを `strlen()` で求め、for 文により配列の要素を 1 文字ずつ 0x20,0x2c,0x2e と比較する。)

5.4 文字列の配列

文字列は char 型の 1 次元配列を用いるが文字列の配列は char 型の 2 次元配列を用いる。

```
char 配列名 [文字列の数][文字列の最大の長さ+1];
```

とする。文字列の配列の初期化は、

```
char 配列名 [][文字列の最大の長さ+1]={ 初期化子並び };
```

とする。

初期化子並びとしては、文字列リテラルを , (カンマ) で並べる。

```
char str[10][40];
```

は、長さ 40 文字 (ナル文字を含む) の文字列が 10 個からなる配列である。char 型の配列としては 2 次元であるが、文字列の配列としては 1 次元である。

3 番目の文字列は str[2] で表す。3 番目の文字列をキーボードから入力するには、

```
fgets(str[2], strlen(str[2], stdin);
```

または、

```
scanf("%s", &str[2]);
```

とし、出力するには

```
puts(str[2]);
```

または、

```
printf("%s", str[2]);
```

とする。前者は出力の後改行されるが後者は改行されない。

例 5.2 例 3.11 の入力と出力部分のプログラム

```
/* jmark.c */
/* 成績の処理 */
#include <stdio.h>
#define N 200 /* 人数 */
#define M 3 /* 科目数 */

int main(void)
{
  int mark[N][M],sum[N]={0};
  int i,j;
  char subj[M][5]={"英語","数学","国語"}; /* 文字列の配列 */

  for(i=0; i<N; i++)
    for(j=0; j<M; j++){
      printf("%d 番の %s の点", i, subj[j]);
      scanf("%d", &mark[i][j]);
      sum[i] += mark[i][j];
  }
  printf("\n");

  printf("%4s %5s %5s %5s %5s\n\n","番号",subj[0],subj[1],subj[2],"合計");
  for(i=0;i<N;i++)
    printf("%4d %5d %5d %5d %5d\n",i,mark[i][0],mark[i][1],mark[i][2],sum[i]);

  return 0;
}
```

演習 5.4 jmark.c において氏名も出力できるように修正してみよ。

演習 5.5 英語で色の名前を入力し、日本語の色の名前を出力するプログラムをつくれ。(“red” と入力すると” 赤” と出力する)

Hint1. 色名前の辞書を文字列配列を用いて表す。2 次元の配列 2 つを使う。

Hint2. 2 つの文字列 str1,str2 が等しいか否かの判定は、`if (strcmp(str1,str2)==0)` とする。(`if (str1==str2)` はだめ。)

文字列の 2 次元配列は char 型の 3 次元配列を用いて実現できる。

演習 5.6 月曜 I 限から金曜 IV 限までの時間割りを 2 次元文字列配列を用いて出力するプログラムをつくれ。

5.5 ポインタ配列

ポインタは配列にすることができる。

```
/* pointer-array.c */
#include <stdio.h>

int main(void)
{
  char *subject[3];

  subject[0]="英語";
  subject[1]="数学";
  subject[2]="国語";

  printf("&subject[0]=%p \n",&subject[0]);
  printf("&subject[1]=%p \n",&subject[1]);
  printf("&subject[2]=%p \n",&subject[2]);
  printf("subject[0]=%p %s\n",subject[0],subject[0]);
  printf("subject[1]=%p %s\n",subject[1],subject[1]);
  printf("subject[2]=%p %s\n",subject[2],subject[2]);

  return 0;
}
```

`char *subject[3]` は、char 型へのポインタ 3 個からなる配列を宣言している。

`subject[0]="英語";` は、文字列"英語"をメモリ上にとり、文字列の先頭アドレスがポインタ `subject[0]` に格納される。

出力結果は以下。

```
$ cc pointer-array.c
$ ./a.out
&subject[0]=0xbffff940
&subject[1]=0xbffff944
&subject[2]=0xbffff948
subject[0]=0x1f70 英語
subject[1]=0x1f78 数学
subject[2]=0x1f80 国語
```

図示すると

アドレス	0xbffff940	0xbffff944	0xbffff948
ポインタ	subject[0]	subject[1]	subject[2]
値	0x1f70	0x1f78	0x1f80

アドレス	0x1f70	0x1f78	0x1f80
値	”英語”	”数学”	”国語”

となる。

なお、pointer-array.c の main() 関数の最初の 5 行はポインタ配列の初期化を用いると

```
char *subject[3]={"英語","数学","国語"};
```

となる。

5.6 main() 関数の引数

main() 関数にコマンドラインから文字列を引数として渡すことができる。この引数のことをコマンドライン引数 (command line argument) という。

```
int main(int argc, char *argv[])
```

argc は入力した文字列の個数、argv[] に入力した文字列を格納する。

```
/* echo.c */
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i;

  printf("argc=%d\n",argc);
  for(i=0;i<argc;i++)
    printf("argv[%d]=%s\n",i,argv[i]);

  return 0;
}
```

出力結果は以下。

```
$ cc echo.c -o echo -Wall
$ ./echo This is a string.
argc=5
argv[0]=./echo
argv[1]=This
argv[2]=is
argv[3]=a
argv[4]=string.
```

実行するコマンド ./echo もコマンドライン引数である。