第11章　「文字列とポインタ」のまとめ

11.1　用語のまとめ

教 p.248

文字列とポインタ　文字列リテラルを評価すると、その文字列リテラルの ____________________________ になる。

```
char str[] = "ABC";    /* 配列 ...  */
char *ptr  = "123";    /* ポインタ ...  */
```

教 p.249

配列とポインタの共通点　添字を使って読み出すときも、違いはない。

教 p.250

配列とポインタの違い　代入をしようとすると違いが現れる。

```
str = "DEF";   /* コンパイル時エラーになる ← 配列は直接代入できない */
ptr = "456";   /* エラーにならない */

str[0] = 'A';   /* エラーにならない、配列の各要素は書き込み可 */
ptr[0] = '4';   /* 結果は処理系により異なるができないと思った方がよい */
```

教 p.252

文字列の配列

```
char st[3][6] = {"Turbo", "NA", "DOHC"};
char *pt[3]   = {"12345", "12", "1234"};
```

どちらも可能だがメモリ中の配置が異なる（Fig.11-5 参照）。

コマンドラインパラメータ　プログラムを起動する時にコマンドラインパラメータを渡すことができる。

コマンドラインパラメータは C プログラムの中では “ポインタで実現する文字列の配列” （__________ 型、あるいは ________ 型）として表現されている。

argvexample.c

```
/* ↓の行は int main(int argc, char **argv) と書いても同じ */
int main(__________, ________________) {
    int i;

    for (i = 0; i <argc; i++) {
        printf("%s¥n", argv[i]);
    }

    return 0;
}
```

argv[0] に“プログラムを起動したコマンド名”が入る。argc にはコマンドラインパラメータの個数（argv[0] も含む）が入る。

実行例: （... の部分は実行環境により異なる。）

```
> argvexample hello! 1 2
... ¥argvexample.exe
hello!
1
2
```

教 p.254

文字列の長さを調べる（ポインタ版） p.216 のプログラムをポインタを使って書き換える。

```
int str_length(__________________) {
    int len = 0;

    while (_____) {
        len++;
    }

    return len;
}
```

- const をつけるとポインタの指すところを書き換えることができなくなる。（const char *は、「char への const なポインタ」ではなくて、「const な char へのポインタ」と解釈される。）
- *s++は、*(s++) と解釈される（教科書 p.177 参照）。
- **（復習）** ポインタをインクリメントすると、1 つ後ろの要素を指すようになる。
- **（復習）** **後置**増分演算子++は、インクリメントされる**前**の値を返す。

教 p.256

文字列のコピー

```
char *str_copy(char *d, const char *s) {
    char  *t = d;

    while (_______________)  /* _____________ */
        ;

    return t;
}
```

- **（復習）**代入式は代入後の左オペランドの値になる（教科書 p.98）。

現在のコンパイラなら、

```
while (d[i] = s[i]) {  /* _____________ */
    i++;
}
```

と書いても効率が極端に悪くなるようなことはない。しかし、前者のようなポインタを使った書き方が簡潔なため好まれる傾向がある。

教 p.258

文字列の不正なコピー　文字列リテラルを書き換えてはいけない。文字列リテラルを書き換えたときの振る舞いは未定義である。

教 p.259

ポインタを返す関数　str_copy 関数が char *を返しているのは、この関数を利用する部分を簡潔に書けるようにするためである。例えば、

```
str_copy(s2, s1);
printf("s2 = %s¥n", s2);
```

を

```
printf("s2 = %s¥n", str_copy(s2, s1));
```

と書くことができる。

教 p.260

文字列操作関数　いくつかの文字列操作関数が string.h ヘッダーに宣言されている。

名前と型	説明
size_t strlen(const char *s)	文字列の長さ
char *strchr(const char *str, int c)	文字列中の文字の出現位置
char *strcpy(char *s1, const char * s2)	文字列のコピー
char *strncpy(char *s1, const char * s2, size_t n)	文字列のコピー*
char *strcat(char *s1, const char *s2)	文字列の連結
char *strncat(char *s1, const char *s2, size_t n)	文字列の連結*
int strcmp(const char *s1, const char * s2)	文字列の比較
int strncmp(const char *s1, const char * s2, size_t n)	文字列の比較*

*のついている関数は操作する文字数に制限をかけることができる。

教 p.260

空ポインタ　“何も指さない” ポインタ定数として _____というマクロが用意されている。NULL はエラーや例外的な状況を表現するのに利用される。NULL は実は定数 0 なので、if 文や while 文の条件式として使用すると _____を意味する。（NULL 以外のポインタは必ず 0 以外、つまり真 (true) になる。）

NULL は stddef.h に定義されている。（**stdio.h**, stdlib.h, string.h, time.h のいずれかをインクルードしても良い。）

教 p.264

文字列変換関数　"144"や"3.14"などの文字列を数値に変換する関数が、stdlib.h に宣言されている。

名前と型	説明
int _____(const char *nptr)	int 型に変換する
long atol(const char *nptr)	long 型に変換する
double _____(const char *nptr)	double 型に変換する