プログラミングⅠ

report#6

提出日：2013/06/27(木)

所属：工学部情報工学科

学籍番号：135713B

氏名：天願 寛之

目次

1. ポインタについての考察

1 ポインタと配列について

1-a-1 ソースコード

```
01 #include <stdio.h>
02
03 int main(){
04     char str[] = "we were child"; /*char 型変数を宣言し、文字列を代入*/
05     char str_p; /*char 型ポインタ変数を宣言*/
06
07     str_p = str; /*ポインタに str を代入*/
08
09     printf("アドレス = %x¥n",&str);
10     printf("保持する文字配列 = %s¥n",str);
11
12     printf("ポインタアドレス = %x¥n", &str_p);
13     printf("保持する値 = %x¥n", str_p);
14     printf("保持するアドレスがもつ内容 = %s¥n",str_p);
15
16     return(0);
17   }
```

1-a-2 実行結果

```
アドレス = 55ee5b70
保持する文字配列 = we were child
ポインタアドレス = 55ee5b68
保持する値 = 55ee5b70
保持するアドレスがもつ内容 = we were child
```

1-a-3 解説

イ. 04 行目で文字配列を変数に格納している。

ロ. 05 行目でポインタ変数を宣言するときには[データ型 *変数名]のようにしている。

ハ. 10 行目、14 行目で変数の持っている文字配列を表示させるため”%s”を用いている。

ニ. 09 行目、12 行目でポインタのアドレスを表示させるために代入するポインタ変数の前に”&”を置いている。”&”はアドレスを取り出すのに用いる事が出来る。

1-a-4 考察

イ. 実行結果よりポインタ変数の持つ値とは格納した変数のアドレスである。実際 07 行目でポインタ変数に”str”と代入しているが”&”を加えて実行しても結局は格納した変数のアドレスを代入することが分かった。

1-a-5 ソースコード

```
#include <stdio.h>

int main(){
  char str[256] = "we were child"; /*char 型変数を宣言し、文字列を代入*/
  char str_p; /*char 型ポインタ変数を宣言*/

  str_p = str; /*ポインタに str のアドレスを代入*/

  printf("アドレス = %x¥n",str);
  printf("保持する文字配列 = %s¥n",str);

  printf("ポインタアドレス = %x¥n", &str_p);
  printf("保持する値 = %x¥n", str_p);
  printf("保持するアドレスがもつ内容 = %s¥n",*str_p);
  return(0);
}
```

1-a-6 解析

イ. このソースコードは 1-a-1 のソースコードの 14 行目でポインタ変数の指すアドレスの内容を表示させるために代入されるポインタ変数の前に”*”を置いたものである

ロ. このコードでは 14 行目の出力の時に”segmentation fault”が出てしまう。

ハ. 14 行目の出力形式を”%s”から”%c”に変えることで’w’だけを表示させる事が出来る。

1-a-7 考察

イ. 14 行目などでの[*ポインタ変数]はポインタの先頭アドレスの中身だけを代入させたことが分かった。

1-b-1 ソースコード

```
01 #include <stdio.h>
02
03 int main(){
04     char str[256] = "we were child"; /*char 型変数配列を宣言し、文字列を代入*/
05     char str_p; /*char 型変数を宣言し、文字列を代入*/
06     int i; /*int 型変数の宣言*/
07     str_p = str; /*ポインタに str のアドレスを代入*/
08
09     printf("アドレス = %x¥n",&str);
10     printf("保持する文字配列 = %s¥n",str);
11
12     printf("ポインタアドレス = %x¥n", &str_p);
13     printf("ポインタの保持する値 = %x¥n", str_p);
14     printf("保持するアドレスがもつ内容 = ");
15     for (i=0; *str_p != NULL; str_p++){ /*for ループ*/
16     printf("%c",*str_p); /*ポインタ str_p の指す配列の要素を表示*/
17     }
18     printf("¥n");
19     return(0);
20 }
```

1-b-2 実行結果

```
アドレス = 5e024a80
保持する文字配列 = we were child
ポインタアドレス = 5e024a78
保持する値 = 5e024a80
保持するアドレスがもつ内容 = we were child
```

1-b-3 解説

イ. 配列やポインタのアドレスは異なるが[*ポインタ変数]を用いて表示させるために作成したコードである。

ロ. 15 行目で for 文を用いてポインタの指すアドレスを 1 つずつずらして、表示させる文字を変えている。

ハ. 16 行目でポインタの指す配列の要素を表示させるために代入するポインタ変数の前に”*”を置き、表示形式を”%c”にしている。

1-b-4 考察

イ. for 文を用いてポインタ変数をインクリメントすることでポインタの指すアドレスをずらすことが出来た。

ロ. ポインタ変数をインクリメントすることで次の要素を出力する事が出来る事から、ポインタ変数のインクリメントによりポインタは次の要素が格納されているアドレスを指す事が分かる。

1-c-1 ソースコード

```
01 #include <stdio.h>
02
03 int main(){
04     char str[256] = "we were child"; /*char 型変数を宣言し、文字列を代入*/
05     char *str_p;
06     int i;
07     str_p = str; /*ポインタに str のアドレスを代入*/
08
09     printf("アドレス = %x¥n",&str);
10     printf("保持する文字配列 = %s¥n",str);
11
12     printf("ポインタアドレス = %x¥n", &str_p);
13     printf("ポインタの保持する値 = %x¥n", str_p);
14     for (i=0; *str_p != NULL; str_p++){ /*for ループ*/
15     printf("%c¥t",*str_p);
16     printf("ポインタの指すアドレス = %x¥n",str_p);
17     }
18     printf("¥n");
19     return(0);
20 }
```

1-c-2 実行結果

```
アドレス = 54ab1b76
保持する文字配列 = we were child
ポインタアドレス = 54ab1b68
ポインタの保持する値 = 54ab1b76
w          ポインタの指すアドレス = 54ab1b76
e          ポインタの指すアドレス = 54ab1b77
           ポインタの指すアドレス = 54ab1b78
w          ポインタの指すアドレス = 54ab1b79
e          ポインタの指すアドレス = 54ab1b7a
r          ポインタの指すアドレス = 54ab1b7b
e          ポインタの指すアドレス = 54ab1b7c
           ポインタの指すアドレス = 54ab1b7d
c          ポインタの指すアドレス = 54ab1b7e
h          ポインタの指すアドレス = 54ab1b7f
i          ポインタの指すアドレス = 54ab1b80
l          ポインタの指すアドレス = 54ab1b81
d          ポインタの指すアドレス = 54ab1b82
```

1-c-3 解析

イ. 実際にポインタの指すアドレスを表示させるプログラムを作成した。

ロ. for 文でポインタの指す値を表示させるついでにポインタの指すアドレスを表示させ、それを繰り返させる。

ハ. アドレスは 16 進数表記させている。

1-c-4 考察

イ. ポインタの保持する値が文字配列の先頭アドレスであることが確認出来る。

ロ. “&”で文字配列を代入格納した char 型変数アドレスを取り出すときは先頭アドレスが取り出されることが確認出来る。

ハ. アドレスを 16 進数表記させていて、スペースを含め 1bite ずつアドレスが変わっていることが分かる。

1-d-1 ソースコード

```
#include <stdio.h>

int main(){
    char str[256] = "we were child"; /*char 型変数を宣言し、文字列を代入*/
    int *str_p; /*int 型ポインタ変数の宣言*/
    int i; /*int 型変数の宣言*/
    str_p = str; /*ポインタに str のアドレスを代入*/

    printf("配列先頭アドレス = %x¥n",&str);
    printf("保持する文字配列 = %s¥n",str);

    printf("ポインタアドレス = %x¥n", &str_p);
    printf("保持する値 = %x¥n", str_p);
    for (i=0; *str_p != NULL; str_p++){ /*for ループ*/
    printf("%c¥t",*str_p);
    printf("ポインタの指す先頭アドレス = %x¥n",str_p);
    }
    printf("¥n");
    return(0);
}
```

1-d-2 実行結果

```
配列先頭アドレス = 58a23b76
保持する文字配列 = we were child
ポインタアドレス = 58a23b68
保持する値 = 58a23b76
w       ポインタの指す先頭アドレス = 58a23b76
e       ポインタの指す先頭アドレス = 58a23b7a
c       ポインタの指す先頭アドレス = 58a23b7e
d       ポインタの指す先頭アドレス = 58a23b82
        ポインタの指す先頭アドレス = 58a23b86
?       ポインタの指す先頭アドレス = 58a23b8a
!       ポインタの指す先頭アドレス = 58a23b8e
?       ポインタの指す先頭アドレス = 58a23b92
        ポインタの指す先頭アドレス = 58a23b96
?       ポインタの指す先頭アドレス = 58a23b9a
        ポインタの指す先頭アドレス = 58a23b9e
?       ポインタの指す先頭アドレス = 58a23ba2
```

1-d-3 解析

イ. 1-c-1 ソースコードのポインタのデータを int 型に書き換えたものである。

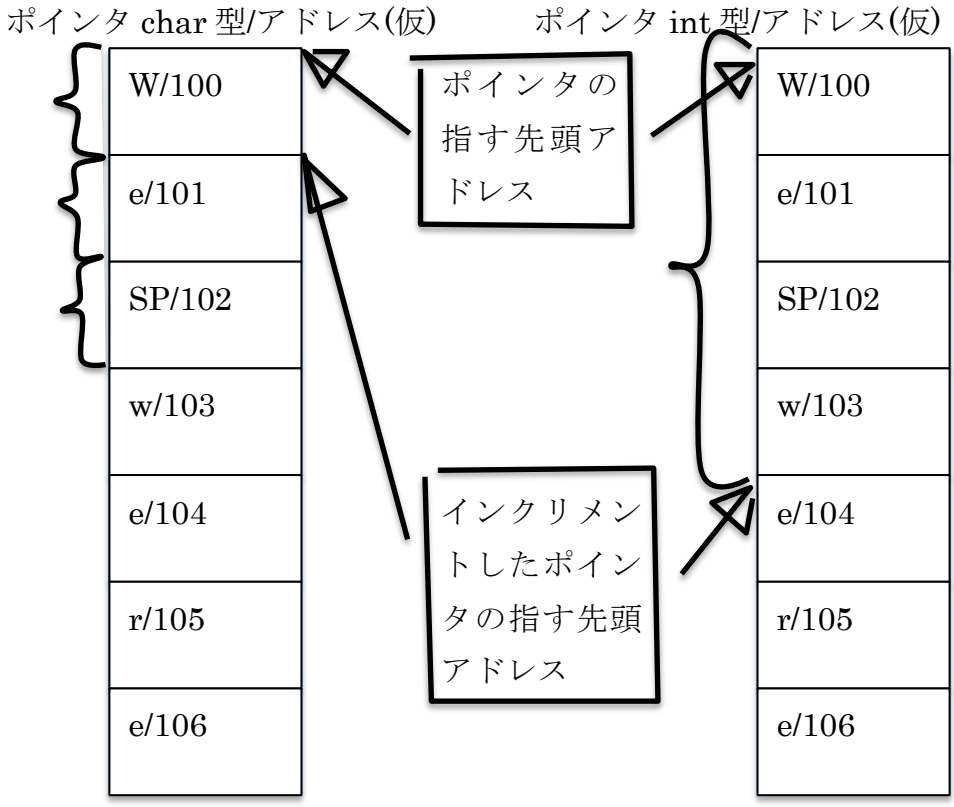
1-d-4 考察

イ. int 型のポインタ変数をインクリメントした値、つまりそのポインタの指すアドレスは 4bite 分のブロックを格納しているので、4bite 先の先頭アドレスを指すことが分かった。

ロ. 実行結果では文字列の最後に格納された”d”以降は”?”が出力されてしまっている。

ハ. 1-c,1-d プログラムで示した概念を以下の 1-d-5 図で表す。

1-d-5 図



1-e-1 ソースコード

```
01 #include <stdio.h>
02
03 int main(){
04   char str[256] = "we were child";/*char 型変数を宣言し、文字列を代入*/
05   char *str_p; /*char 型ポインタ変数を宣言*/
06   char **str_pp; /*char 型２重ポインタ変数を宣言*/
07
08   str_p = str; /*ポインタに str を代入*/
09   str_pp = &str_p; /*ポインタのアドレスを２重ポインタに代入*/
10
11   printf("配列先頭アドレス = %x¥n",str);
12   printf("保持する文字配列 = %s¥n",str);
13
14   printf("ポインタアドレス = %x¥n", &str_p);
15   printf("保持する値 = %x¥n", str_p);
16   printf("保持するアドレスがもつ内容 = %s¥n",str_p);
17
18   printf("保持する値 = %x¥n",str_pp);
19   printf("保持する値が持つ値 = %x¥n",*str_pp);
20   printf("保持する値が持つ値の内容 = %s", *str_pp);
21
22   return(0);
23 }
```

1-e-2 実行結果

```
配列先頭アドレス = 598ceb76
保持する文字配列 = we were child
ポインタアドレス = 598ceb68
保持する値 = 598ceb76
保持するアドレスがもつ内容 = we were child
保持する値 = 598ceb68
保持する値が持つ値 = 598ceb76
保持する値が持つ値の内容 = we were child
```

1-e-3 解析

イ．このソースコードは２重ポインタの役割を示す為に作成したものである。

ロ．２重ポインタ変数の宣言には[データ型**変数名]と記述する。

1-e-4 考察

イ．２重ポインタが保持する値はポインタのアドレスであることが分かる。

ロ．19 行目で[*２重ポインタ変数]で表示させている値、つまり２重ポインタの指す値が持つ値は文字配列の先頭アドレスと分かる。さらにその内容も示す事ができている。(printf 文などで代入させる時の”*”の役割は同じということ。)

ハ．これらから２重ポインタの役割はポインタを指しポインタのアドレスを格納することがわかる。

ニ．よって[**２重ポインタ]で表示させることが出来る内容はその２重ポインタが指すポインタが指す値の内容であることが予測出来る。

1-e-5 ソースコード

```
01 #include <stdio.h>
02
03 int main(){
04   char str[] = "we were child"; /*char 型変数を宣言し、文字列を代入*/
05   char *str_p; /*char 型ポインタ変数を宣言*/
06   char **str_pp; /*char 型２重ポインタ変数を宣言*/
07   int i; /*int 型変数の宣言*/
08   str_p = str; /*ポインタに str のアドレスを代入*/
09   str_pp = &str_p; /*２重ポインタに str_p のアドレスを代入*/
10
11   printf("配列先頭アドレス = %x¥n",str);
12   printf("保持する文字配列 = %s¥n",str);
13
14   printf("ポインタアドレス = %x¥n", &str_p);
15   printf("保持する値 = %x¥n", str_p);
16   printf("保持するアドレスがもつ内容 = %s¥n",str_p);
17
18   printf("保持する値が持つ値 = %x¥n",*str_pp);
19   for (i=0; *str_p != NULL; str_p++){
20     printf("%c¥t",**str_pp);
21     printf("２重ポインタの指すアドレス = %x¥n",str_pp);
22   }
23   return(0);
24 }
```

1-e-6 実行結果

```
配列先頭アドレス = 5c00eb76
保持する文字配列 = we were child
ポインタアドレス = 5c00eb68
保持する値 = 5c00eb76
保持するアドレスがもつ内容 = we were child
保持する値が持つ値 = 5c00eb76
w          2重ポインタの指すアドレス = 5c00eb68
e          2重ポインタの指すアドレス = 5c00eb68
           2重ポインタの指すアドレス = 5c00eb68
w          2重ポインタの指すアドレス = 5c00eb68
e          2重ポインタの指すアドレス = 5c00eb68
r          2重ポインタの指すアドレス = 5c00eb68
e          2重ポインタの指すアドレス = 5c00eb68
           2重ポインタの指すアドレス = 5c00eb68
c          2重ポインタの指すアドレス = 5c00eb68
h          2重ポインタの指すアドレス = 5c00eb68
i          2重ポインタの指すアドレス = 5c00eb68
l          2重ポインタの指すアドレス = 5c00eb68
d          2重ポインタの指すアドレス = 5c00eb68
```

1-e-7 解析

イ. 20行目の[**２重ポインタ]で2重ポインタが指すポインタが指す値の内容を表示させることを示すために作成したソースコードである。

1-e-8 考察

イ. 予想通りの結果が出せた。

2. コマンドラインから受け取った文字列の大文字と小文字を変換するプログラムを作成せよ。入力は1バイトの表示文字とし、アルファベット文字以外は変換しない。

2-a ソースコード

```
01 #include <stdio.h>
02
03 void replace(char *, char *); /*関数のプロトタイプ宣言*/
04 char *dest, *str; /*char 型ポインタ変数のグローバル宣言*/
05
06 int main(int argc, char **argv){
07   int i; /*int 型整数の宣言*/
08
09   for (i=0; *argv != NULL; i++,argv++){ /*for ループ*/
10     printf("parameter(%2d)¥t%s¥n",i,*argv); /*i とポインタ argv の指す値を代入*/
11     replace(dest,*argv); /*関数 replace の呼び出し(dest と*argv を引数)*/
12     printf("¥n"); /*改行*/
13   }
14   return(0);
15 }
16
17 /*関数 replace の定義*/
18 void replace(char *dest, char *str){
19
20   while (*str != '¥0'){ /*while ループ*/
21     if('A' <= *str && *str <= 'Z'){ /*小文字なら実行*/
22       dest = *str + 32; //大文字に変換後 dest に代入
23     }else if('a' <= *str && *str <= 'z'){ /*大文字なら実行*/
24       dest = *str - 32; /*小文字に変換後 dest に代入*/
25     }else{ /*条件式以外なら実行*/
26       dest = *str; /*変換せずに dest に代入*/
27     }
28
29     printf("%c",dest); /*dest の値を代入*/
30     str++; /*ポインタ str のインクリメント*/
31   }
32 }
```

2-b 実行結果

```
This is sample STRING Asm#6

parameter( 1)       This
tHIS
parameter( 2)       is
IS
parameter( 3)       sample
SAMPLE
parameter( 4)       STRING
string
parameter( 5)       Asm#6
aSM#6
```

2-c 解析

イ. コマンドラインから受け取った文字列の大文字と小文字を変換するプログラムを作成した。

ロ. int main に続く”()”の中に引数として”int argc, char **argv”を入れているが、これはコマンドラインパラメータと言い、出力する際に入力する文字列を引数として受け取る仕組みになっている。

ハ. 11 行目の”replace 関数”の呼び出しで引数として” dest,*argv”を入れているがこの”*argv”は 18 行目の”replace 関数”の定義で”char *str”として扱っている。”dest”は”char *dest”である。

ニ. 20 行目から 27 行目までが while 文と if-else 文を用いた大文字小文字の判定で、1 文字 1 文字大文字は小文字に、小文字は大文字に、それ以外はそのまま、22,24,26 行目で dest に代入され、29 行目で出力されている。

ホ. 30 行目でインクリメントし続け、￥0(改行)が出力されるまでループしている。

2-d 考察

イ. 12 行目で“This”の後一旦￥0(改行)が出力されたため 09 行目のループに戻ったことからコマンドラインで入力した文字列をスペースで分けてポインタ変数 argv がよみとっていると分かる。

ロ. しかし、09 行目のループではポインタのインクリメントでコマンドラインを全て読み取っていることが分かる。

3. 文字列を反転して表示するプログラムも作成せよ。（例 ″abcd″ => ″dcba″）

3-a. ソースコード

```
01 #include <stdio.h>
02
03 void replace(char *, char *);
04 char *dest, *str;
05
06 int main(int argc, char **argv){
07   int i;
08
09   for (i=0; *argv != NULL; i++,*argv++) {
10     printf("parameter(%2d)¥t%s¥n",i,*argv);
11     replace(dest,*argv);
12     printf("¥n");
13   }
14   return(0);
15 }
16
17 void replace(char *dest, char *str){
18   int num, num2;
19     for (num=0; *str != '¥0'; str++,num++);
20   for (num2=0; num2 <= num; str--,num2++){
21   dest = *str;
22   printf("%c",dest);
23     }
24 }
```

3-b. 実行結果

```
This is sample STRING Asm#6

parameter( 1)       This
sihT
parameter( 2)       is
si
parameter( 3)       sample
elpmas
parameter( 4)       STRING
GNIRTS
parameter( 5)       Asm#6
6#msA
```

3-c. 解析

イ. 文字列を反転して表示するプログラムを作成した。

ロ. 18 行目で int 型変数 num と num2 を宣言している。

ハ. 19 行目の for 文でループしている。num の初期値 0、ループ条件*str の値が NULL 文字以外の間、ループ毎に str と num をインクリメントする。

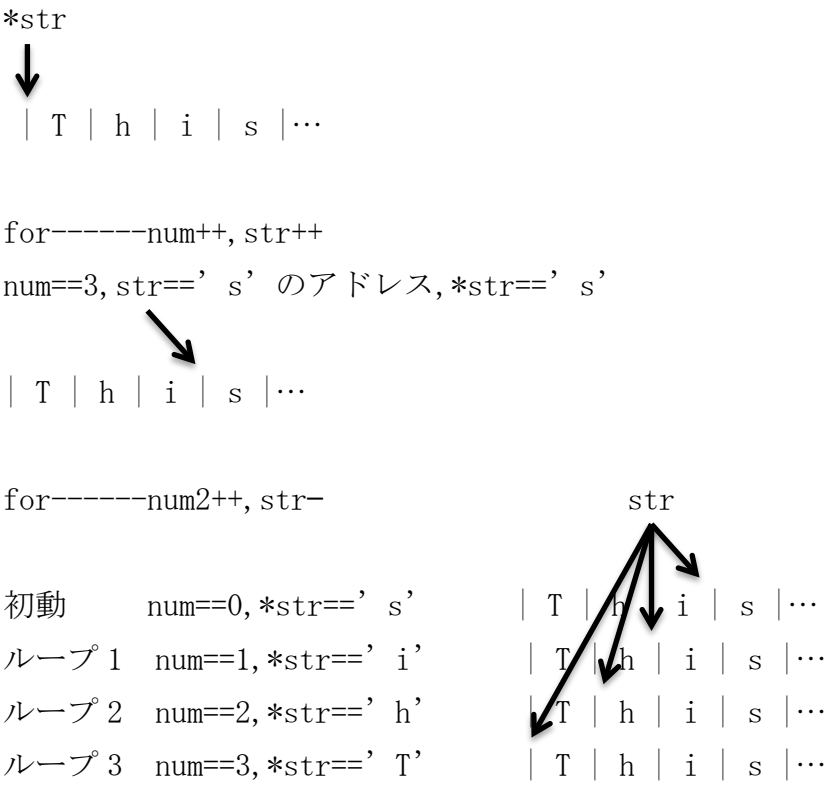
ニ. 20 行目の for 文でループしている。num2 の初期値 0、ループ条件 num2 が num 以下の間、ループ毎に str をデクリメント、num2 をインクリメントする。さらに dest に*str を代入し、printf 関数で dest の値を出力している。

3-d. 考察

イ. 19 行目で*str の値が NULL 文字が出るまで str をインクリメントすることにより*str の値が入力した文字列の最後の文字になる。そして num の値はその移動した文字数の値、つまり入力した文字数-1 になる。

ロ. 20 行目で num2 の値が num、つまり入力した文字数－１になるまでインクリメントし、同時に str をデクリメントしながら 1 つ１つ出力することにより、文字列の最後の文字からの出力を行っている。

ハ. 具体的な動作を入力”This”で図に示す。



4. 反省・感想

最後は全然時間が足りてないことに気づいた。次からはもっと計画的にやりたい。