プログラミング Ⅰ

Report#6

提 出 日:2006 年某日(木)

所 属 :工学部情報工学科

学籍番号: 065702G

氏名 : 新 垣 智 規

問題 a.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。

ソースコード(change.c)

```
/*                                                                    Program   : change2.c
Commennts : アルファベット大文字小文字入れ替えプログラム。                    */

#include <stdio.h>

int get_n(char *);
void print_data(char *,int);
void replace(char *dest, char *str);



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

  printf("入力された文字列数 = %d\n",argc);
  printf("文字列のアドレス = %08x\n\n",&argc);

  for(i = 1,argv++; *argv!= NULL; i++, argv++){
    printf("列番号(%2d)\t%s\n",i,*argv);
    printf("'%s' ==>文字変換実行==>'",*argv);
    replace(*argv,*argv);
    printf("'");
    printf("\n");
    n=get_n(*argv);
    printf("\n");
    print_data(*argv,n);
    printf("文字列の先頭の値 = %c\n",**argv);
    printf("文字列の収容されているアドレス = %08x\n",*argv);
```

```
    printf("ポインタのアドレス = %08x\n",argv);
    printf("argv-*argv = %08x\n",(long)argv-(long)*argv);
    printf("\n");
  }
  return(0);
}

void replace(char *dest, char *str){
  int i;
  for(i=0; str[i]!= NULL; i++){

    if(65 <= str[i]&&str[i] <= 90){
      dest[i] = (str[i]+32);
      printf("%c",dest[i]);
    }else if(97 <= str[i]&&str[i] <=122){
    dest[i] = (str[i]-32);
    printf("%c",dest[i]);
    }else{
    dest = (str);
    printf("%c",dest[i]);
  }
  }

}



int get_n(char *pa){
  int i;
  for (i=1; *pa != '\0'; i++,pa++);
  return(i);
}

void print_data(char *pa,int n){
```

```
  printf("文字列の文字数=%2d %s\n",n,pa);
}
```

◎実行結果

実行結果 1.「a bcde fg hij k」の変換結果。

```
[tomonori-arakaki-no-ibook-g4:~] j06002% ./change2 a bcde fg hij k
入力された文字列数 = 6
文字列のアドレス = bffffab8

列番号( 1)      a
'a' ==>文字変換実行==>'A'

文字列の文字数= 2 A
文字列の先頭の値 = A
文字列の収容されているアドレス = bffffbee
ポインタのアドレス = bffffb54
argv-*argv = ffffff66

列番号( 2)      bcde
'bcde' ==>文字変換実行==>'BCDE'

文字列の文字数= 5 BCDE
文字列の先頭の値 = B
文字列の収容されているアドレス = bffffbf0
ポインタのアドレス = bffffb58
argv-*argv = ffffff68

列番号( 3)      fg
'fg' ==>文字変換実行==>'FG'

文字列の文字数= 3 FG
文字列の先頭の値 = F
```

文字列の収容されているアドレス = bffffbf5

ポインタのアドレス = bffffb5c

argv-*argv = ffffff67

列番号(4)　　hij

'hij' ==>文字変換実行==>'HIJ'

文字列の文字数= 4 HIJ

文字列の先頭の値 = H

文字列の収容されているアドレス = bffffbf8

ポインタのアドレス = bffffb60

argv-*argv = ffffff68

列番号(5)　　k

'k' ==>文字変換実行==>'K'

文字列の文字数= 2 K

文字列の先頭の値 = K

文字列の収容されているアドレス = bffffbfc

ポインタのアドレス = bffffb64

argv-*argv = ffffff68

実行結果 2.「We Are Studying Programing Very Hard!」の変換

[tomonori-arakaki-no-ibook-g4:~] j06002% ./change2 We Are Studying Programing Very Hard!

入力された文字列数 = 7

文字列のアドレス = bffffa78

列番号(1)　　We

'We' ==>文字変換実行==>'wE'

文字列の文字数= 3 wE

文字列の先頭の値 = w

文字列の収容されているアドレス = bffffbc6

ポインタのアドレス = bffffb28

argv-*argv = ffffff62

列番号(2)　　Are

'Are' ==>文字変換実行==>'aRE'

文字列の文字数= 4 aRE

文字列の先頭の値 = a

文字列の収容されているアドレス = bffffbc9

ポインタのアドレス = bffffb2c

argv-*argv = ffffff63

列番号(3)　　Studying

'Studying' ==>文字変換実行==>'sTUDYING'

文字列の文字数= 9 sTUDYING

文字列の先頭の値 = s

文字列の収容されているアドレス = bffffbcd

ポインタのアドレス = bffffb30

argv-*argv = ffffff63

列番号(4)　　Programing

'Programing' ==>文字変換実行==>'pROGRAMING'

文字列の文字数=11 pROGRAMING

文字列の先頭の値 = p

文字列の収容されているアドレス = bffffbd6

ポインタのアドレス = bffffb34

argv-*argv = ffffff5e

列番号(5)　　Very

'Very' ==>文字変換実行==>'vERY'

文字列の文字数= 5 vERY

文字列の先頭の値 = v

文字列の収容されているアドレス = bffffbe1

ポインタのアドレス = bffffb38

argv-*argv = ffffff57

列番号(6)　　Hard!

'Hard!' ==>文字変換実行==>'hARD!'

文字列の文字数= 6 hARD!

文字列の先頭の値 = h

文字列の収容されているアドレス = bffffbe6

ポインタのアドレス = bffffb3c

argv-*argv = ffffff56

◎考察

ポインタを表示しながら、文字の大小変換をするプログラム。
プログラムの構造はまず、関数を指定している。
このプログラムでは「int get_n(char *);」「void print_data(char *,int);」「void replace(char *dest, char *str);」という３つの関数と、main 関数で構成している。
まず、main 関数内で printf を使い入力された文字列の個数、そのアドレスを表示する。
その後、for 文を用いて繰り返し処理をする。
この繰り返し処理とは、入力された文字列を１つ１つ処理していくという意味で、「./a.out 文字列α　文字列β　文字列γ」と入力した場合は文字列α→文字列β→文字列γの順番でスペースで区切られた（NULL コードに達した）文字列を処理していくという内容である。「i++」と入力されて

いるので、繰り返されるごとに i の値が１づつ増えていく。
繰り返し処理する内容は、「列番号の表示」→「replace()の実行（文字の大小変換の実行）」→「get_n()の戻り値を n に代入（文字列の文字数をカウント）」→「n を使い print_data()を実行（n を表示する）」→「各変数のポインタを表示」というもので、文字列全ての処理が終わるまで続く。
ここで、「void replace(char *dest, char *str);」の説明をする。
この関数内の処理を簡潔に述べると、文字列の大小変換。
入力された文字列をＩつ１つ処理していくという関数で NULL コードに達したら処理を終了する。つまり「NiGht」と入力された文字列を処理すると「N」→「i」→「G」→「h」→「t」の順に変換処理が繰り返されるという意味である。
replace()内で使われている if 文は入力された文字が小文字なら大文字に、大文字なら小文字に変換、それ以外はそのまま出力するというもの。
このプログラムもそうだが、アドレスの理解には NULL コードの存在を頭に入れた上で、どう組み立てていくかが重要である。文字列の区切りには NULL コードが利用されているため、このようなプログラムを利用するときには非常に重要である。

問題 b.また、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"）

◎ソースコード

```
/*
        Program   : change3.c
        Comments : 文字列反転プログラム
*/

#include <stdio.h>
```

```
int get_n(char *);
void print_data(char *,int);
void replace(char *dest, char *str);

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

  printf("入力された文字列数 = %d\n",argc);
  printf("文字列のアドレス = %08x\n\n",&argc);

  for(i = 1,argv++; *argv!= NULL; i++, argv++){
    printf("列番号(%2d)\t%s\n",i,*argv);
    replace(*argv,*argv);
    printf("\n");
    n=get_n(*argv);
    printf("\n");
    print_data(*argv,n);
    printf("文字列の先頭の値 = %c\n",**argv);
    printf("文字列の収容されているアドレス = %08x\n",*argv);
    printf("ポインタのアドレス = %08x\n",argv);
    printf("argv-*argv = %08x\n",(long)argv-(long)*argv);
    printf("\n");
  }
  return(0);
}
void replace(char *dest, char *str){
  int i;
  for(i=1,str++; *str!= NULL; i++,str++){
  }
  printf("反転実行 =");
  for(; i>=0; str--,i--){
    dest = str;
```

```
    printf("%c",*dest);
  }
}
```

```
int get_n(char *pa){
  int i;
  for (i=1; *pa != '\0'; i++,pa++);
  return(i);
}

void print_data(char *pa,int n){
  printf("文字列の文字数=%2d %s\n",n,pa);
}
```

◎実行結果

実行結果 1.「I have no money.」の変換

```
[tomonori-arakaki-no-ibook-g4:~] j06002% ./change3 I have no money.
入力された文字列数 = 5
文字列のアドレス = bffffab8

列番号( 1)      I
反転実行 =I

文字列の文字数= 2 I
文字列の先頭の値 = I
文字列の収容されているアドレス = bffffbee
```

ポインタのアドレス = bffffb58

argv-*argv = ffffff6a

列番号(2)　　have

反転実行 =evah

文字列の文字数= 5 have

文字列の先頭の値 = h

文字列の収容されているアドレス = bffffbf0

ポインタのアドレス = bffffb5c

argv-*argv = ffffff6c

列番号(3)　　no

反転実行 =on

文字列の文字数= 3 no

文字列の先頭の値 = n

文字列の収容されているアドレス = bffffbf5

ポインタのアドレス = bffffb60

argv-*argv = ffffff6b

列番号(4)　　money.

反転実行 =.yenom

文字列の文字数= 7 money.

文字列の先頭の値 = m

文字列の収容されているアドレス = bffffbf8

ポインタのアドレス = bffffb64

argv-*argv = ffffff6c

実行結果 2.「I want to sleep.」の変換

[tomonori-arakaki-no-ibook-g4:~] j06002% ./change3 I want to sleep.

入力された文字列数 = 5

文字列のアドレス = bffffab8

列番号(1)　　I

反転実行 =I

文字列の文字数= 2 I

文字列の先頭の値 = I

文字列の収容されているアドレス = bffffbee

ポインタのアドレス = bffffb58

argv-*argv = ffffff6a

列番号(2)　　want

反転実行 =tnaw

文字列の文字数= 5 want

文字列の先頭の値 = w

文字列の収容されているアドレス = bffffbf0

ポインタのアドレス = bffffb5c

argv-*argv = ffffff6c

列番号(3)　　to

反転実行 =ot

文字列の文字数= 3 to

文字列の先頭の値 = t

文字列の収容されているアドレス = bffffbf5

ポインタのアドレス = bffffb60

argv-*argv = ffffff6b

列番号(4)　　sleep.

反転実行 =.peels

文字列の文字数= 7 sleep.

文字列の先頭の値 = s

文字列の収容されているアドレス = bffffbf8

ポインタのアドレス = bffffb64

argv-*argv = ffffff6c

◎ 考察

入力された文字列を、逆にして返すというプログラム。
逆というのは「abcd」を「dcba」にするといったものである。
全体の流れとしては、問題 a と同じなので割合。
内容としては、replace 関数の中身を変更したのみ。
問題 b.での replace の構造はまず、NULL コードまで i を増やす、つまり文字列の最後の文字までアドレスを移動させる。最初の for 文の処理はこのことを意味する。
つまりこの replace 関数はまず文字列の最後にアドレスを飛ばし、その後の for 文で i—を使い文字列の先頭文字に向かって処理をする、という関数。この場合の処理は、表示するということ。よってこの関数では文字列の最後の文字から最初の文字に向かって出力されていく、という仕組みになっている。

◎ポインタの考察

上記の２つのプログラムで、各々ポインタを使いアドレスを書き出している。
そこでまず気付くのが「ポインタのアドレス」の規則性。
処理を行うごとに 4bit づつ増えているのが上記のプログラムの実行結果から分かる。つまり、ポインタのアドレスは１つに付き 4bit の容量を確保している、と言える。
次に目につくのが「文字列の収容されているアドレス」。
これは、その文字列の収容されているアドレス最初のアドレスである。
今回の実行結果を良く見てみると、「文字列の収容されているアドレス」＋「文字列の文字数」＝　次の「文字列の収容されているアドレス」となっていることに気付く。

例を挙げてみる。

[例]　change3.c
[tomonori-arakaki-no-ibook-g4:~]　j06002%　./change3　abc defghi
入力された文字列数 = 3
文字列のアドレス = bffffab8

列番号(1)　　abc
反転実行 =cba

文字列の文字数= 4 abc
文字列の先頭の値 = a
文字列の収容されているアドレス = bffffbf6
ポインタのアドレス = bffffb68
argv-*argv = ffffff72

列番号(2)　　defghi
反転実行 =ihgfed

文字列の文字数= 7 defghi
文字列の先頭の値 = d
文字列の収容されているアドレス = bffffbfa
ポインタのアドレス = bffffb6c
argv-*argv = ffffff72

change3.c の上記の結果を例に取ってみる。
ここで列番号 1 の「文字列の収容されているアドレス」は 「bffffbf6」。
そして文字数は 4 である。(NULL コードを含む)
次に、列番号 2 の「文字列の収容されているアドレス」を見てみると「文字列の収容されているアドレス = bffffbfa」とある、

ここで「bffffbfa = bffffbf6 + 4 」は成り立っている。
つまり、文字列の収容されているアドレスは続いて確保されていることが分かる。
最後に「ポインタのアドレス」とあるが、これも規則性があり「4bit」ずつ増えていることが分かる。（例以外の実行結果でも証明されている）
これはつまり、アドレスの為に確保されるメモリ容量は、4bit である、という意味を表している。

◎ 感想・反省
難しくなってきましたプログラミング・・・、最初の頃のレポートを難しいと言っていた自分を殴り飛ばしたいです、でも今難しく思えることもこのまま勉強していったら基本なんだろうな。。。
プログラムを勉強した友達に話を聞くと「ポインタさえ理解出来ていれば、他のところで遅れていても大丈夫。むしろここで出来るか出来ないかが分かれる」と聞かされていたので自分なりに努力をしてみました。
といっても、完璧に理解はしていないと思います。
感覚的には分かっているのに、細かいところの説明ができないという不思議現象が起きていて、理解しようと躍起になってましたが完璧にはほど遠いでしょう。C の基本＝ポインタ、だからこそ一番重要な所。これを理解しないと先に進めない、という話も聞きました。
何事も基本が大事ってことで、何度も参考書や HP を読み返し、完璧に仕上げたいと思います。