プログラミング I

REPORT#6

提出日	:	2012年 7月12日　木曜日
所属	:	琉球大学工学部情報工学科
学籍番号	:	125722G
氏名	:	知念 辰明

1.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。

a-1:ポインタ*src,*dest 及びサブルーチン replace を使用したプログラム。

高水準コードの全体▼

```
/*
  program   :proto.c
  student-ID:125722G
  author    :Tatsunori CHINEN
  date      :07/07/2012 (SAT)
  comment   :lower => upper and upper => lower with pointer.
 */
#include <stdio.h>
#include <stdlib.h>

void replace(char *str,char *dest);                     //サブルーチン宣言
void counter(char *argv);
int n;                                                  //int型変数nを宣言

int main(int argc,char **argv){                         //メイン関数開始

    char *str,*dest;                                    //char型ポインタstr,dest宣言
    while(*argv != NULL){                               //*argvに格納されている値がNULLであれば繰り返す

        puts("\n====phase1==boost");                    //----フェイズ1,準備段階
        counter(*argv);                                 //"*argv"が示す文字を先頭とする文字列の文字数をnにカウント
        dest = (char *)malloc(n);                       //ポインタ"dest"に,"n"の値分確保したメモリのアドレスを入力
        if (*dest == NULL){                             //ポインタ"dest"内のアドレスがNULLでないかを確認
            puts("Memory allocation was failed.");
            return(0);                                  //確保されていなければプログラムを終了。
        }
        puts("Memory allocation was finished.");

        puts("====phase2==midcourse");                  //----フェイズ2,メインプログラム,変化適用
        str = *argv;                                    //ポインタstrに*argvに格納されているアドレスを代入
        replace(str,dest);                              //サブルーチンreplaceにポインタstrとdestを渡す
        puts("Sub routine was finished");

        puts("====phase3==terminal");                   //----フェイズ3,結果表示
        printf("str[%s] => dest[%s]\n",str,dest);       //変更される前とされた後の文字列を出力。
        free((char *)dest);
        argv++;                                         //ポインタargvをインクリメントし,次の*argvを示す

    }
    return(0);                                          //プログラムを終了

}

counter(char *argv){                                    //メインから受け取ったアドレスをポインタ*argvで受け取る
    n = 0;                                              //ファイル内共通変数nを0で初期化
    while(*argv != NULL){                               //*argvがNULLで無ければ繰り返す
        n = n + 1;                                      //nに1を足す
        argv++;                                         //argvをインクリメントし,文字をずらす。
    }
}

replace(char *str,char *dest){                  //メインから受け取った2つのアドレスをポインタ*str,*destで受け取る

    int i = 0;                                          //int型変数iを宣言し0で初期化
    while(str[i] != NULL){                              //str[i]がNULLでなければ繰り返す
        if                          (islower(str[i])){ //str[i]に格納されている値が小文字であれば分岐
             dest[i] = toupper(str[i]);                 //-------str[i]を大文字にした値をdest[i]に格納
        }else  if                   (isupper(str[i])){ //str[i]に格納されている値が大文字であれば分岐
             dest[i] = tolower(str[i]);                 //-------str[i]を少文字にした値をdest[i]に格納
        }else dest[i] = str[i];                         //どちらでもなければそのままの値をdest[i]に格納
        i++;                                            //変数iをインクリメント
    }

}
```

コマンドラインへ入力したコマンド及びパラメータ▼ ※実行可能ファイル名は"a.out"である。

```
./a.out Sample Program
```

実行結果▼

```
====phase1==boost
Memory allocation was finished.
====phase2==midcourse
Sub routine was finished
====phase3==terminal
str[./a.out] => dest[./A.OUT]

====phase1==boost
Memory allocation was finished.
====phase2==midcourse
Sub routine was finished
====phase3==terminal
str[Sample] => dest[sAMPLE]

====phase1==boost
Memory allocation was finished.
====phase2==midcourse
Sub routine was finished
====phase3==terminal
str[Program] => dest[pROGRAM]
```

考察▼

・コマンドラインパラメータから,ポインタ"**argv"のデータを"*str"で受け取り,"*dest"へ転写するプログラムを作成した。
・コマンドラインパラメータとして,コマンド名である"./a.out"も文字列として処理している。
・23 行目"str = *argv"において,ポインタ"str"に入力されているのは"*argv"に格納されているアドレスである。
・ポインタ"*str"および"*dest"を宣言しているが,"*str"にアクセスしようとすると"segmentation fault"が表示される。
・一方で"*dest"については,21 行目 malloc 関数が実行された後であればアクセスが可能になる。
・malloc()関数は,サイト『初心者のためのポイント学習 C 言語』によれば,()内の値分メモリを確保し確保したメモリの先頭アドレスを返す関数であるという。失敗した場合は"NULL"を返す。
・malloc()関数で確保されたメモリは,使用されたあと free()関数で解放される。

a-2:ポインタとアドレスの挙動の観察。

高水準コードの一部▼

```
8   #include <stdio.h>
9
10  int main(int argc, char **argv){
11      char *str;
12
13      for                                             (argv++; *argv != NULL; argv+=1){
14          printf("\nCharacters ==> %10s ( *argv =%08x)\n",*argv,*argv);
15          str = *argv;
16          for                                         (number=0; *str != NULL; str++){
17              printf("Character  ==> %10c (  *str =%08x)\n",*str,&*str);
18          }
19      }
20      return(0);
21
22  }
```

コマンドラインへ入力したコマンド及びパラメータ▼ ※実行可能ファイル名は"a.out"である。

```
./a.out KISARAGI attention
```

実行結果▼

```
Characters ==>   KISARAGI ( *argv =677d2c73)
Character  ==>          K (  *str =677d2c73)
Character  ==>          I (  *str =677d2c74)
Character  ==>          S (  *str =677d2c75)
Character  ==>          A (  *str =677d2c76)
Character  ==>          R (  *str =677d2c77)
Character  ==>          A (  *str =677d2c78)
Character  ==>          G (  *str =677d2c79)
Character  ==>          I (  *str =677d2c7a)

Characters ==>  attention ( *argv =677d2c7c)
Character  ==>          a (  *str =677d2c7c)
Character  ==>          t (  *str =677d2c7d)
Character  ==>          t (  *str =677d2c7e)
Character  ==>          e (  *str =677d2c7f)
Character  ==>          n (  *str =677d2c80)
Character  ==>          t (  *str =677d2c81)
Character  ==>          i (  *str =677d2c82)
Character  ==>          o (  *str =677d2c83)
Character  ==>          n (  *str =677d2c84)
```

考察▼

・コマンドラインから得られたパラメータを,文字列と文字として,それぞれのアドレスと共に出力するプログラムを作成した。
・プログラム内では char 型ポインタ"*str"を宣言しているが,これは 2 重ポインタではインクリメントによる各メモリに割り振られた 1 文字ずつの移動が行えなかった為である。

・char 型ポインタ"str"を宣言せずに同様の動作をさせるには,
インクリメントではなく以下のコードの 14 行目の様な"+1"という動作をさせなければならなかった。

高水準コードの一部▼

```
8   #include <stdio.h>
9
10  int main(int argc, char **argv){
11
12      for                                             (argv++; *argv != NULL; argv+=1){
13          printf("\nCharacters ==> %10s ( *argv =%08x)\n",*argv,*argv);
14          for                                         (number=0; **argv != NULL; argv+=1){
15              printf("Character  ==> %10c ( *argv =%08x)\n",**argv,&**argv);
16          }
17      }
18      return(0);
19
20  }
```

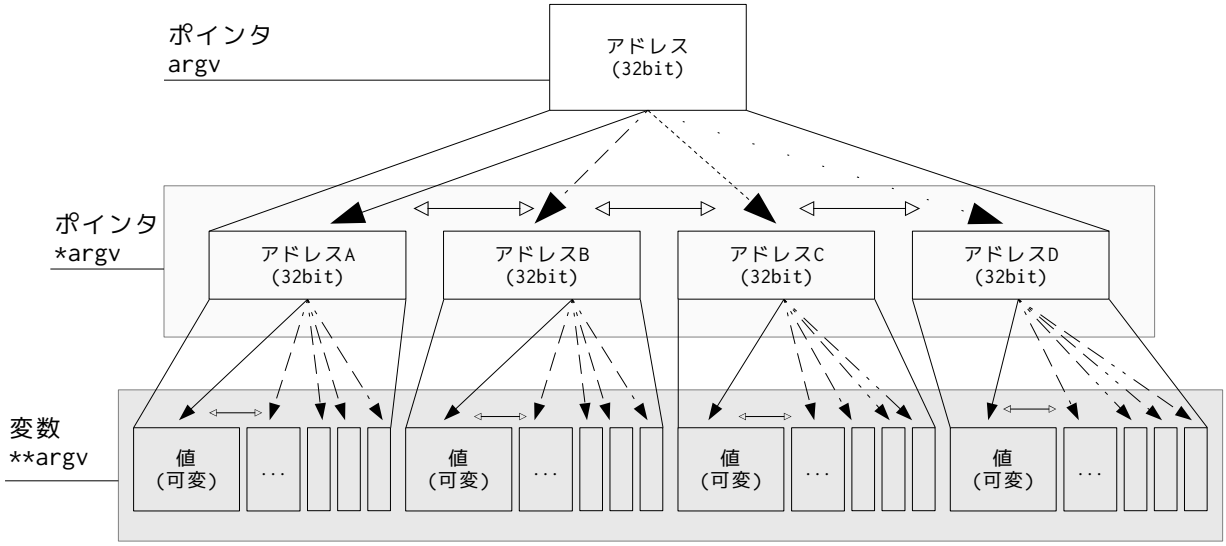
・インクリメント(argv++)のままで実行した場合,次のような出力が得られた。

実行結果▼

```
Characters ==>   KISARAGI ( *argv =6c41fc73)
Character  ==>          K ( *argv =6c41fc73)
Character  ==>          a ( *argv =6c41fc7c)
zsh: segmentation fault  ./sample.o KISARAGI attention
```

・次の文字列の先頭文字へ移動してしまっている。
・"segmentation fault"というエラーが発生している。

・以上のことより,次のことが確認できた。



・変数を宣言する際にアスタリスク"*"を付けると,その変数はポインタとして宣言される。
・"**argv"について考えると,アスタリスクが 2 つ付いていることからいわゆる"二重ポインタ",上図のような構造になっている。
・図上で最も上位に存在するポインタ"argv"は,メモリ上のポインタ"*argv"として扱うメモリのアドレスとして初期状態ではアドレス A が格納されているメモリを指している。
・このとき"*argv"はある値が格納されているメモリのアドレスを格納している。

・"argv"をインクリメントすると,"argv"に格納されているアドレスの値が（増加量は 1 に限らない）増加する。
・結果,"*argv"として扱われるメモリのアドレスが変更され,それに伴って"**argv"の値も変化する。
・ポインタが複数重なっているこの場合,"*argv"をインクリメントすると示す値は次の文字ではなく次の文字列の先頭文字である。
・"*argv"の値をインクリメントではなく"+1"で処理するか,別の二重以上でないポインタにアドレスを渡すことでこれらの動作を避け,同じ文字列内で次の文字を参照することが出来る。

・このような挙動から,複数の文字をあたかも文字列であるかのように扱うことができる。
ある配列に格納されている文字を文字列として処理したいとき,文字列としてサブルーチンに渡したいときなど大きな効果を発揮する。

a-3:"segmentation fault"と malloc()関数の観察。

高水準コードの一部▼

```
8   #include <stdio.h>
9   #include <stdlib.h>
10
11  int main(){
12      char *str;
13      printf("Before str(P) = %08x\n",str);
14      str = malloc(5);
15      if ( str == NULL ) return(0);
16      printf("After  str(P) = %08x\n",str);
17      free((char *)str);
18      printf("Last   str(P) = %08x\n",str);
19      return(0);
20  }
```

実行結果▼

```
Before str(P) = 00000000
After  str(P) = 0bd008b0
Last   str(P) = 0bd008b0
```

考察▼

・次にエラー"segmentation fault"と malloc()関数について観察したい。
・a-1 項のプログラム 22 行目では,ポインタ"*dest"に対して malloc()関数でメモリを確保しているが,
この処理が無ければ"*dest"についてもアクセスの際に"segmentation fault"が発生する。
・Wikipedia"セグメンテーション違反"の項によれば,アクセスが許可されていないメモリへアクセス,
あるいは許可されていない方法でアクセスした際に表示されるものであるらしい。
・作成したプログラムから得られた結果として,malloc 関数を使う前のポインタ"str"が示すアドレスは
"00000000"であり,この状態で"*str"にアクセスしようとすると以下のようなエラーが得られる。

高水準コードの一部▼

```
8   #include <stdio.h>
9   #include <stdlib.h>
10
11  int main(){
12      char *str;
13      printf("Before str(P) = %08x\n",str);
14      printf("       str(C) = %8c\n",*str);
15      str = malloc(5);
16      if ( str == NULL ) return(0);
17      printf("After  str(P) = %08x\n",str);
18      free((char *)str);
19      printf("Last   str(P) = %08x\n",str);
20      return(0);
21  }
```

実行結果▼

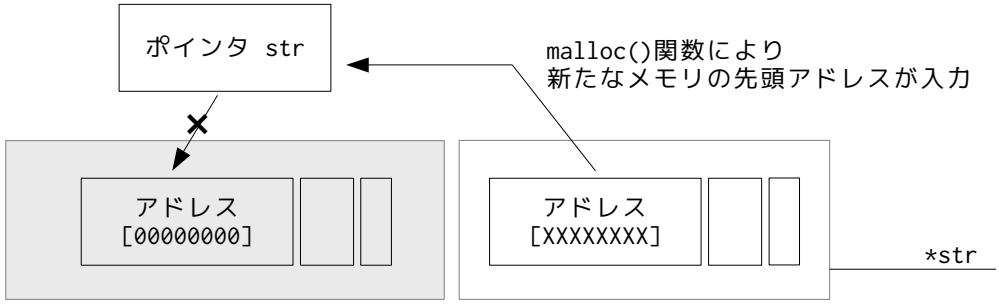
```
Before str(P) = 00000000
zsh: segmentation fault  ./a.out
```

・一方で,malloc()関数でポインタ"str"に確保したメモリのアドレスを入力した後では通常通りアクセスが
可能となる。
・上記コード 14 行目,"   printf("   str(C) = %8c\n",*str);"を切り取り
17 行目と 18 行目の間に挿入したプログラムの実行結果が以下である。

実行結果▼

```
Before str(P) = 00000000
After  str(P) = 016008b0
       str(C) =
Last   str(P) = 016008b0
```

・文字や値は入力されていないものの,アクセスすることができた。
・これらのことより,malloc()関数は以下のような挙動であることが分かる。



アクセス禁止領域(segmentation fault)

2.コマンドラインから受け取った文字列を反転して表示するプログラムを作成せよ。

a-1:ポインタ*src,*dest 及びサブルーチン reverse を使用したプログラム。

高水準コードの全体▼

```
1   /*
2     program   :proto2.c
3     student-ID:125722G
4     author    :Tatsunori CHINEN
5     date      :07/10/2012 (SAT)
6     comment   :Reverse Characters with pointer.
7    */
8   #include <stdio.h>
9   #include <stdlib.h>
10
11  void reverse(char *str,char *dest);                     //サブルーチン宣言
12  void counter(char *argv);
13  int n;                                                  //int型変数nを宣言
14
15  int main(int argc,char **argv){                         //メイン関数開始
16
17      char *str,*dest;                                    //char型ポインタstr,dest宣言
18      argv++;
19      while(*argv != NULL){                               //*argvに格納されている値がNULLであれば繰り返す
20
21          puts("\n====phase1==boost");                   //----フェイズ1,準備段階
22          counter(*argv);                                 //"*argv"が示す文字を先頭とする文字列の文字数をnにカウント
23          dest = (char *)malloc(n);                       //ポインタ"dest"に,"n"の値分確保したメモリのアドレスを入力
24          if (*dest == NULL){                             //ポインタ"dest"内のアドレスがNULLでないかを確認
25              puts("Memory allocation was failed.");
26              return(0);                                  //確保されていなければプログラムを終了
27          }
28          puts("Memory allocation was finished.");
29
30          puts("====phase2==midcourse");                  //----フェイズ2,メインプログラム,変化適用
31          str = *argv;                                    //ポインタstrに*argvに格納されているアドレスを代入
32          reverse(str,dest);                              //サブルーチンreplaceにポインタstrとdestを渡す
33          puts("Sub routine was finished");
34
35          puts("====phase3==terminal");                  //----フェイズ3,結果表示
36          printf("str[%s] => dest[%s]\n",str,dest);       //変更される前とされた後の文字列を出力。
37          free((char *)dest);
38          argv++;                                         //ポインタargvをインクリメントし,次の*argvを示す
39
40      }
41      return(0);                                          //プログラムを終了
42
43  }
44
45  counter(char *argv){                                    //メインから受け取ったアドレスをポインタ*argvで受け取る
46      n = 0;                                              //ファイル内共通変数nを0で初期化
47      while(*argv != NULL){                               //*argvがNULLで無ければ繰り返す
48          n = n + 1;                                      //nに1を足す
49          argv++;                                         //argvをインクリメントし,文字をずらす。
50      }
51  }
52
53  reverse(char *str,char *dest,int n){      //メインから受け取った2つのポインタと変数をポインタ*str,*destで受け取る
54
55      dest[n] = NULL;                                     //文字列の最後にNULLを入力
56      dest = dest + n - 1;                                //destにNULL直前のアドレスを入力
57      while(*str != NULL){                                //"*str"の値がNULLでなければ繰り返す
58          *dest = *str;                                   //*destに*strの値を転写
59          str++;dest--;                                   //ポインタstrはインクリメント,destはデクリメント
60      }
61
62  }
```

コマンドラインへ入力したコマンド及びパラメータ▼ ※実行可能ファイル名は"a.out"である。

```
./a.out BUMP OF CHICKEN
```

実行結果▼

```
====phase1==boost
Memory allocation was finished.
====phase2==midcourse
Sub routine was finished
====phase3==terminal
str[BUMP] => dest[PMUB]

====phase1==boost
Memory allocation was finished.
====phase2==midcourse
Sub routine was finished
====phase3==terminal
str[OF] => dest[FO]

====phase1==boost
Memory allocation was finished.
====phase2==midcourse
Sub routine was finished
====phase3==terminal
str[CHICKEN] => dest[NEKCIHC]
```

考察▼

・コマンドラインパラメータから,ポインタ"**argv"のデータを"*str"で受け取り,"*dest"へ反転して
転写するプログラムを作成した。
・このプログラムでは,コマンドラインから受け取る１つ目のパラメータ"./a.out"をスキップして出力している。

X.2つ目のパラメータによって出力を変更するプログラムを作成。

高水準コードの一部▼

```
#include <stdio.h>
#include <stdlib.h>                         //ヘッダファイルstdlib.hを読み込み
void replace(char *str,char *dest);         //サブルーチン宣言
void counter(char *argv);
void reverse(char *str,char *dest,int n);
int n;                                      //int型変数nを宣言
int main(int argc,char **argv){             //メイン関数開始

    char *str,*dest;                        //char型ポインタstr,dest宣言
    int number = 0,switch_2=0;              //int型変数number,switch_2を宣言し"0"で初期化

    *argv = argv[1];                        //*argvに2つ目のパラメータの先頭アドレスを入力
    if              ( **argv == 'r'){       //2つ目のパラメータの先頭文字が"r"であれば真分岐
        *argv+=1;                           //*argvに入力されているアドレスを増やし,**argvが指す文字をずらす
        if          ( **argv == 'e'){       //2つ目のパラメータの2文字目が"e"であれば真分岐
            *argv+=1;                       //*argvに入力されているアドレスを増やし,**argvが指す文字をずらす
            if      ( **argv == 'p'){       //2つ目のパラメータの3文字目が"p"であれば真分岐
                *argv+=1;
                if    ( **argv ==NULL){
                    switch_2 = 1;           //変数switch_2に"1"を入力
                    *argv = argv[2];        //*argvに3つ目のパラメータの先頭アドレスを入力
                    number = 2;             //変数numberに"2"を入力
                }else
                    switch_2 = 0;           //変数switch_2に"0"を入力
            }else
                switch_2 = 0;               //変数switch_2に"0"を入力
        }else
            switch_2 = 0;                   //変数switch_2に"0"を入力
    } else {
        switch_2 = 0;                       //変数switch_2に"0"を入力
    }
    if              ( switch_2 == 0 )       //switch_2の値が0であれば真分岐
        number = 1;                         //変数numberに"1"を入力
        *argv = argv[number];               //*argvに2つ目のパラメータの先頭アドレスを入力
    }
    while( *argv != NULL ){                 //*argvに格納されているアドレスが"NULL"でなければ繰り返す
        puts("\n====phase1==boost");        //----フェイズ1,準備段階
        counter(*argv);                     //サブルーチンcounterに引数として*argvを渡し呼び出し
        dest = (char *)malloc(n);           //char型のメモリをn個確保し,その先頭アドレスをdestに入力
        if (dest == NULL){                  //destのアドレスが"NULL"であれば真分岐
            puts("Memory allocation was failed.");
            return(0);                      //プログラムを終了
        }
        puts("Memory allocation was finished.");

        puts("====phase2==midcourse");      //----フェイズ2,変化適用
        str = *argv;                        //ポインタstrに*argvに格納されているアドレスを入力
        if ( switch_2 == 1) replace(str,dest);  //switch_2の値によってサブルーチンにポインタstrとdestを渡す
        else                reverse(str,dest,n);//偽分岐であればサブルーチンreverseを呼び出し
        puts("Sub routine was finished");

        puts("====phase3==terminal");
        printf("str[%s] => dest[%s]\n",str,dest);//変更適用前と後の文字列を出力
        free((char *)dest);                 //destに格納されているアドレスのメモリを解放
        number++;                           //numberをインクリメント
        *argv = argv[number];               //*argvが示すアドレスをnumberに伴って変更
    }
    return(0);                              //プログラムを終了

}
void replace(char *str,char *dest){         //メインから受け取った2つのアドレスを*strと*destで受け取る

    while(*str != NULL){                                //*strが"NULL"でなければ繰り返す
        if                      (islower(*str)){    //*strの値が小文字であれば真分岐
            *dest = toupper(*str);                      //----*destに*strを大文字にした値を入力
        }else  if               (isupper(*str)){    //*strの値が大文字でなければ真分岐
            *dest = tolower(*str);                      //----*destに*strを小文字にした値を入力
        }else *dest = *str;                             //いずれも偽であれば*destにそのまま*strを入力
        str++;dest++;                                   //str,destをインクリメント
    }
    *dest = NULL;                                       //*destの文字最後にNULLを入力
}                                                       //===>文字の大文字小文字を反転するサブルーチン
void counter(char *argv){                   //メインから受け取ったアドレスをポインタ*argvで受け取る
    n = 0;                                  //変数nを"0"で初期化
    while(*argv != NULL){                   //*argvのアドレスが"NULL"でなければ繰り返す
        n = n + 1;                          //nの値を"1"増やす
        argv++;                             //argvをインクリメントし,*argvが示す文字をずらす
    }
}                                           //===>文字列の文字数をカウントするサブルーチン
void reverse(char *str,char *dest,int n){   //メインから受け取った2つのアドレスと値を受け取る
    dest[n] = NULL;                         //dest[n]が指す部分に"NULL"を入力
    dest = dest + n - 1;                    //destに格納されているアドレスを,NULLの直前に変更
    while(*str != NULL){                    //*strの値が"NULL"でなければ繰り返す
        *dest = *str;                       //*destに*strの文字を転写
        str++;dest--;                       //strをインクリメントし,destをデクリメント
    }
}
```

コマンドラインへ入力したコマンド及びパラメータ▼ ※実行可能ファイル名は"a.out"である。

```
/a.out  REPLACE REVERSE
```

```
/a.out rep REPLACE REVERSE
```

実行結果▼

```
====phase1==boost
Memory allocation was finished.
====phase2==midcourse
Sub routine was finished
====phase3==terminal
str[REPLACE] => dest[ECALPER]

====phase1==boost
Memory allocation was finished.
====phase2==midcourse
Sub routine was finished
====phase3==terminal
str[REVERSE] => dest[ESREVER]
```

```
====phase1==boost
Memory allocation was finished.
====phase2==midcourse
Sub routine was finished
====phase3==terminal
str[REPLACE] => dest[replace]

====phase1==boost
Memory allocation was finished.
====phase2==midcourse
Sub routine was finished
====phase3==terminal
str[REVERSE] => dest[reverse]
```

考察▼

・コマンドラインから受け取ったパラメータの内,2 つ目の文字列に応じて文字列への処理を変更するプログラムを作成した。
・実行可能ファイルを起動する"./a.out"(ファイル名は任意)のあと,2 つ目の引数として「rep」という文字列があれば,3 つ目以降の引数について大文字と小文字を入れ替える処理を行う。
・2 つ目の引数に「rep」という文字列が無かったり,「rep***」(*は任意の文字)のように「rep」の以降に文字が続いているとき,2 つ目以降の引数の文字順を入れ替える処理を行う。
・例として,「reps」という引数を 2 つ目に入力した際の実行結果を示す。

```
====phase1==boost
Memory allocation was finished.
====phase2==midcourse
Sub routine was finished
====phase3==terminal
str[reps] => dest[sper]

====phase1==boost
Memory allocation was finished.
====phase2==midcourse
Sub routine was finished
====phase3==terminal
str[REPLACE] => dest[ECALPER]

====phase1==boost
Memory allocation was finished.
====phase2==midcourse
Sub routine was finished
====phase3==terminal
str[REVERSE] => dest[ESREVER]
```

XXX.あとがき。（反省・感想・参考）

a-1.参考サイト・文献。

・『C 実践プログラミング 第三版』（オーム社）
・『Open Lecture』
  http://www.osn.u-ryukyu.ac.jp/lecture/wiki/index.php?Open%20Lecture
・『初心者のためのポイント学習 C 言語』
  http://www9.plala.or.jp/sgwr-t/lib/malloc.html
・『Wikipedia-セグメンテーション違反』
  http://ja.wikipedia.org/wiki/セグメンテーション違反

a-2.反省・感想。

ポインタ(*str など)を宣言したあと,すぐに"*str"などにアクセスしてしまって
"segmentation fault"が多発したのが一番の壁でした…。
ポインタとして宣言した時点では,アドレスを格納する部分しかメモリが確保されていないということを
これから念頭に入れてプログラミングを行いたいと思います。

コマンドラインからの入力が行えるようになったことで,更にできることが増えました。
C 言語を学ぶ上で中核となるというポインタを,今回と次回のレポートでしっかりと理解していきたいです。

最後までご覧頂きありがとうございました。