プログラミングⅠ

report#4

提出日：2013 年 5 月 30 日(木)

所属：琉球大学工学部情報工学科

学籍番号：135713B

氏名：天願寛之

目次

1. 標準ライブラリ関数 islower(),toupper()を使い、下記の trlowup プログラムを書き換えて新規に trupper プログラムを作成せよ。

1-a-1. ソースコード

```
01 /*
02   Program  : trupper.c
03   Comments : translate lower case characters into upper case ones.
04 */
05 /*header file*/
06 #include <stdio.h>
07 #include <ctype.h>
08
09
10 char trupper(char);/*関数 trupper のプロトタイプ宣言*/
11
12 int main(){
13  char c;/*入力した文字を格納する変数の宣言*/
14
15  while( (c=getchar()) != EOF )/*EOF でループを抜け出す*/
16    putchar( trupper(c) );/*関数 trupper の呼び出し*/
17  return(0);
18 }
19 /*関数 trupper*/
20 char trupper( char c ){
21   if (islower(c))
22    return(toupper(c));
23   else
24     return( c );
25 }
```

1-a-2. 出力結果(2 行ずつ表記してあるが上段が入力した値である。以後同じ)

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ

trupperKANSUU
TRUPPERKANSUU

123da--!!(*~*)
123DA--!!(*~*)
```

1-a-3. 解析

イ. “islower()”は()内に代入された文字が小文字であれば真(0 以外の数)を返し、代入された文字が英小文字以外なら偽(0)を返す。

ロ. “toupper()”は()内に代入された文字が小文字であれば大文字に変換した値を返し、小文字以外ならそのまま値を返す。

ハ. ヘッダーファイルに大文字、小文字などの文字検査や大文字から小文字への文字変換などを行うために”ctype.h”を読み込ませる。

ニ. 16 行目で関数 trupper を呼び出す為に、10 行目でプロトタイプ宣言をする。

ホ. 20 行目から関数 trupper 定義についての記述を行う。

ヘ. “getchar()”は標準入力(キーボード)から 1 文字入力して、’c’に代入

し、”putchar()”で’trupper(c)’を標準出力に出力する。その際、間違った入力がされた場合は’EOF’を出力し、ループを抜け出す。
ト. “islower()”は” ‘a’ <= c && c <= ’z’ “と同様の働きをしている。
チ. “toupper()”は” c-‘a’+’A’ “と同様の働きをしている。

1-a-4. 考察
イ. 標準ライブラリ関数とは異なる、自分で作成したユーザ関数を使うことが出来るとようになる事が分かった。
ロ. “islower()”と”toupper()”を調べてサンプルプログラムと比較することで同様な意味をなす部分を見つけた。
ハ. “関数 main”の変数と”関数 trupper”の変数は異なっていても正常にコンパイルされた。
ニ. 1 の解析通り、”islower()”を” ‘a’ <= c && c <= ‘z’ “に、”toupper()”を” c-‘a’+’A’ “にいずれかを置き換えても正常にコンパイルされた。
ホ. “getchar()”は 1 文字の入力が行われることから文字列を入力するときの動きを観察する。

1-b. “getchar()”の動作の観察

1-b-1. ソースコード

```
01 #include <stdio.h>
02 #include <ctype.h>
03
04 char trupper(char);/*関数 trupper のプロトタイプ宣言*/
05
06 int main(){
07   char c;/*入力した文字を格納する変数の宣言*/
08
09   while( (c=getchar()) != EOF )/*EOF でループを抜け出す*/
10     {  printf("!");
11        putchar( trupper(c) );/*関数 trupper の呼び出し*/
12     } return(0);
13 }
14 /*関数 trupper*/
15 char trupper( char c ){
16   if (islower(c))
17     return(toupper(c));
18   else
19     return( c );
20 }
```

1-b-2. 実行結果

```
i am you
!I! !A!M! !Y!O!U!
```

1-b-3. 解析

イ. while 文の中に printf 文を加えて出力の流れをみた。

1-b-4. 考察

ロ. ”getchar()”は入力された文字列を 1 文字ずつ’c’に代入し、それを 1 文字ずつ”putchar(trupper(c))”が出力している流れが見える。

2. trupper プログラムを書き換えて、rot13 暗号化・複合プログラムを作成せよ。

rot13 とは、次のようにアルファベットを 13 文字ずらす暗号化の方法である。

```
A ==> N   a ==> n
B ==> O   b ==> o
C ==> P   c ==> p
….        ….
Z ==>M    z ==>m
```

2-a-1. 暗号化プログラムソースコード

```
01 /*header file*/
02 #include <stdio.h>
03 #include <ctype.h>
04
05 char rot13(char);/*関数 rot13 のプロトタイプ宣言*/
06
07 int main(){
08   char c;/*入力した文字を格納する変数の宣言*/
09   puts(“encryption”);
10   while( (c=getchar()) != EOF )/*EOF でループを抜け出す*/
11     putchar( rot13(c) );/*関数 rot13 の呼び出し*/
12   return(0);
13 }
14 /*関数 rot13*/
15 char rot13( char c ){
16   if ('A' <= c && c <= 'M' | 'a' <= c && c <= 'm')
17     return(c+13);
18   if ('N' <= c && c <= 'Z' | 'n' <= c && c <= 'z')
19     return(c-13);
20   return(c);
21 }
```

2-a-2. 実行結果

```
encryption
abcdefghijklmNOPQRSTUVWXYZ
nopqrstuvwxyzABCDEFGHIJKLM

ABCDEFGHIJKLMnopqrstuvwxyz
NOPQRSTUVWXYZabcdefghijklm

123da----(*~*)===>{p_q}
123qn----(*~*)===>{c_d}
```

2-a-3. 解析

イ. ユーザ関数名を”rot13”に変更し、11 行目で呼び出し、15 行目以降で定義している。

ロ. 15 行目以降の“関数 rot13”の if 文で、入力された文字が文字’A’以上’M’以下、’a’以上’m’以下の時は入力した文字に 13 を足して返し、文字’N’以上’Z’以下、’n’以上’z’以下の時は入力した文字から 13 を引いて返し、それ以外はそのまま返している。

ハ. 文字の足し引きなどの演算は ASCⅡコード表での演算と同価値である。

ニ. 文字列を書き込む関数である”puts”を用いて文字列”encryption”を出力した。

2-a-4. 考察

イ. サンプルプログラムを参考にして範囲条件の元でいくつずらすかを指定する事で作成することが出来た。

ロ. “islower()”と”toupper()”を使用していないのでヘッダの ctype.h を使用せずに出力したら 2-a と同様にコンパイルされた。

2-b-1. 複合プログラムソースコード

```
01 /*header file*/
02 #include <stdio.h>
03
04 char rot13(char);/*関数 rot13 のプロトタイプ宣言*/
05
06 int main(){
07   char c;/*入力した文字を格納する変数の宣言*/
08   puts("decryption");
09   while( (c=getchar()) != EOF )/*EOF でループを抜け出す*/
10     putchar( rot13(c) );/*関数 rot13 の呼び出し*/
11       return(0);
12 }
13 /*関数 rot13*/
14 char rot13( char c ){
15   if ('A' <= c && c <= 'M' | 'a' <= c && c <= 'm')
16     return(c+13);
17   if ('N' <= c && c <= 'Z' | 'n' <= c && c <= 'z')
18     return(c-13);
19   return(c);
20 }
```

2-b-2. 実行結果

```
decryption
nopqrstuvwxyzABCDEFGHIJKLM
abcdefghijklmNOPQRSTUVWXYZ

NOPQRSTUVWXYZabcdefghijklm
ABCDEFGHIJKLMnopqrstuvwxyz

123qn----(*~*)===>{c_d}
123da----(*~*)===>{p_q}
```

2-b-3. 解析

イ. 関数 puts で出力する文字列を”decryption”に変えただけである。

2-b-4. 考察

イ. rot13 の暗号化は大文字小文字 a~z の文字列、全部で 26 文字の中、前半部の 13 文字を順に 13 文字ずらすと後半部の文字が順に現れ、また、その逆も同じく前半部の文字が順に現れることになるので対応する文字が対となっている。従って、2-a で暗号化された文字を入力すれば、元の文字に変換されることになるのである。

ロ. 単純な暗号化は複合化も簡単に行えた。

3. オリジナルの暗号化・複合プログラムを作成せよ。

3-a-1. 暗号化ソースコード

```
01 /*header file*/
02 #include <stdio.h>
03
04 /*関数のプロトタイプ宣言*/
05 char F(char);
06
07 int main(){
08   char x;/*入力した文字を格納する変数の宣言*/
09   puts("encryption");
10   while( (x=getchar()) != EOF )/*EOF でループを抜け出す*/
11     putchar( F(x) );/*関数の呼び出し*/
12   return(0);
13 }
14 /*関数 F*/
15 char F( char x ){
16   if (0x21 <= x && x <= 0x4f)
17     return(x+47);
18   if (0x50 <= x && x <= 0x7e)
19     return(x-47);
20   if (x == 0x20)
21     return(x);
22   return(x);
23 }
```

3-a-2. 実行結果

```
abcdefghijklmNOPQRSTUVWXYZ
23456789:;<=>}~!"#$%&'()*+

ABCDEFGHIJKLMnopqrstuvwxyz
pqrstuvwxyz{|?@ABCDEFGHIJK

123da----g(*^*)p!!!?
`ab52¥¥¥¥BWY/YXAPPPn
```

3-a-3. 解析

イ．キーボードから入力できる文の中で、入力した値が 16 進数の'7e'ならそのまま出力し、'21'~'4f'までの値には 47 を足し、'50'~'7e'までの値は 47 を引いた値が出力されるようにした。

3-a-4. 考察

イ．これは rot13 を応用し、 キーボードで入力出来るスペース()を除いた ASCⅡコード表の 94 文字を前半の 47 文字には 47 を足し、後半の 47 文字からは 47 を引いた値が出るようになっているので、rot13 と同様に対応する文字が対となっている。従って、複合プログラムもこのプログラムを用いることが出来るので今回は複合プログラムを記述しない。

ロ．もっと複雑な暗号化プログラムの作成を試みる。

3-b-1. 暗号化ソースコード

```
01 /*header file*/
02 #include <stdio.h>
03
04 /*関数のプロトタイプ宣言*/
05 char G(char);
06
07 int main(){
08  char x;/*入力した文字を格納する変数の宣言*/
09  puts("encryption");
10   while( (x=getchar()) != EOF )/*EOF でループを抜け出す*/
11     putchar( G(x) );/*関数の呼び出し*/
12   return(0);
13 }
14 /*関数 G*/
15 char G( char x ){
16         if(x == 0x7c)
17         return('!');
18        else if(x == 0x7e)
19         return('”');
20        else if(x % 5 == 0)
21         return(x);
22        else if(x % 5 == 1)
23         return(x+1);
24        else if(x % 5 == 2)
25         return(x+2);
26        else if(x % 5 == 3)
27         return(x+3);
28        else if(x % 5 == 4)
29         return(x+4);
30
31         return(0);
32  }
```

3-b-2. 実行結果

```
abcdefghijklmnopqrstuvwxyz
cegdfhjlikmoqnprtvsuwy{xz|

ABCDEFGHIJKLMNOPQRSTUVWXYZ
ACEGIFHJLNKMOQSPRTVXUWY[]Z

123456789~|(*^*)?#$%&'!
52468:79;”!(,b,*B#%')+$
```

3-b-3. 解析

イ. 20行から29行まででASCⅡコードのキーボードから入力できる文字の範囲の 5 で割った余りが 0 の時はそのまま返し、余りが 1 の時は 1 を足し、2 の時は 2 を、3 の時は 3 を、4 の時は 4 を足して返した。

ロ. 5 で割った余りを足したら入力出来る文字の範囲外に出てしまう’|’と’~’はそれぞれ、’スペース( )’、’!’で返すようにした。

3-b-5. 複合ソースコード

```
01 /*header file*/
02 #include <stdio.h>
03
04 /*関数のプロトタイプ宣言*/
05 char G(char);
06
07 int main(){
08   char x;/*入力した文字を格納する変数の宣言*/
09   puts("encryption");
10   while( (x=getchar()) != EOF )/*EOF でループを抜け出す*/
11     putchar( G(x) );/*関数の呼び出し*/
12   return(0);
13 }
14 /*関数 G*/
15 char G( char x ){
16         if(x == 0x21)
17         return('|');
18        else if(x == 0x22)
19         return('~');
20        else if(x % 5 == 0)
21         return(x);
22        else if(x % 5 == 1)
23         return(x-3);
24        else if(x % 5 == 2)
25         return(x-1);
26        else if(x % 5 == 3)
27         return(x-4);
28        else if(x % 5 == 4)
29         return(x-2);
30         return(0);
31  }
```

3-b-6. 実行結果

```
cegdfhjlikmoqnprtvsuwy{xz|
abcdefghijklmnopqrstuvwxyz

ACEGIFHJLNKMOQSPRTVXUWY[]Z
ABCDEFGHIJKLMNOPQRSTUVWXYZ

52468:79;”!(,b,*B#%')+$
123456789~|(*^*)?#$%&'!
```

3-b-7. 解析

イ. ’スペース()’と’!’はそれぞれ’|’と’~’にもどるようにした。

ロ. 5 で割って余りのないものはそのままで、余りが 1 のものは 1 が足されていたのでその値は 5*n+2(n は不定数)になっていることをふまえて、余りが 2 である値の場合は 1 を引いた値が返されるようにした。この考えで全ての値を元に戻している。

3-b-8. 考察

イ. この暗号化なら対応する文字が対にならずに済むし、変わらない文字や変わる文字が混在しており、少し複雑化しているように感じられる。

ロ. しかし、’A’~’Z’などのように続いた文字を入力すると一定の周期で変わらない文字を出力していることから、実際問題単純な暗号化の位階を超えられていない。

ハ. 思いがけないことによく使われる’A’,’U’,’i’がいずれも変わらず出力されていることが分かるのでこの 3 文字に関して、単独で異なるずらしを行ってみる。

3-b-9. 変更を行った暗号化ソースコードの一部

```
00 /*関数 G*/
01 char G( char x ){
02         if(x == 0x7c)
03          return('!');
04        else if(x == 0x7e)
05          return('”');
06        else if(x == 0x41)
07          return('i');
08        else if(x == 0x55)
09          return('#');
10        else if(x == 0x69)
11          return('U');
12        else if(x % 5 == 0)
13          return(x);
14        else if(x % 5 == 1)
15          return(x+1);
16        else if(x % 5 == 2)
17          return(x+2);
18        else if(x % 5 == 3)
```

```
19          return(x+3);
20        else if(x % 5 == 4)
21          return(x+4);
22          return(0);
23  }
```

3-b-10. 実行結果

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
iCEGIFHJLNKMOQSPRTVX#WY[]Z

abcdefghijklmnopqrstuvwxyz
cegdfhjlUkmoqnprtvsuwy{xz|
```

3-b-11. 解析

イ．元の暗号化コードに下記の部分を足した。

```
      else if(x == 0x41)
        return('i');
       else if(x == 0x55)
        return('#');
       else if(x == 0x69)
        return('U');
```

ロ．'A'は'i'、'U'は'#'、'i'は'U'で返される。

3-b-12. 考察

イ．やはり変わらない文字であり、さらに母音である文字の不定期な出現で分かりにくくなった気がする。

ロ．複合化に関しては被っている訳ではないので単独でずらした値を逆の変換を行えばよい。

ハ．しかし、コードが大きくなることは気になる点である。

ニ．もっと簡素なプログラムを書きたかった。

4. 感想・反省

オリジナルの暗号化に関して言えば、もっと色々な入出力関数を用いて、スマートで複雑なプログラムを書きたかった。しかし色々な入出力関数を調べても暗号化のプログラムにうまく用いる事が出来なかったことが今回の反省である。とは言え、暗号化はとても難しいという思いがあったので今回簡単ではあるが暗号化を完成させることが出来て本当に良かった。

これからの課題で今回の経験を生かしていきたい。