1. sample#1.c を解析し、ASCII コード(0x00～0x7f)の各範囲(Scope)を判断するプログラムを作成せよ。
   範囲例) 数字：figures、英大文字：capital letter、英小文字：small letter、
           非表示文字：Not Printable character、その他：misc.等

＊ sample#1.c：while文、if,if-else,if-else if-else文、真(TRUE)と偽(FALSE)

```
 1  #include <stdio.h>
 2
 3  #define FALSE 0
 4  #define TRUE  !FALSE
 5
 6  int main(){
 7      int  value,c, count;
 8      char line[128];
 9
10      count = 0;
11
12      while(TRUE){
13        count++;
14        if(count > 5) break;
15
16        printf("Enter a HexValue ===> ");
17        fgets(line, sizeof(line), stdin);
18        sscanf(line, "%x", &c);
19
20        printf("Colum=%02d:%%d(%3d)-%%x(%2x)",count,c,c);
21        if(0x20 <= c && c <= 0x7e)
22          printf("-%%c(%c)\n",c);
23        else
24          printf("-Not Printable character\n");
25
26        if      (0x20 <= c && c <= 0x2f){
27          puts("====> Scope_A\n");
28        }else if(0x30 <= c && c <= 0x39){
29          puts("====> Scope_B\n");
30        }else if(0x3a <= c && c <= 0x40){
31          puts("====> Scope_C\n");
32        }else if(0x41 <= c && c <= 0x5a){
33          puts("====> Scope_D\n");
34        }else{
35          puts("====> Scope_E\n");
36        }
37      }
38
39      return(0);
40  }
```

解　析

3行目 ： 特別なテキストエディタでグローバルにFALSEを0に変更。

4行目 ： 3行目と同様に、TRUEをFALSEの否定形に変更。

7行目 ： int型変数、value(未使用)、c(16進数の値を格納)、count(ループ回数)を定義。

8行目 ： char型の配列line[128]を定義。
127文字(文字列の終端記号\0もカウントするため)格納可。

10行目 ： int型変数countに0を代入(初期化)。

12-37行目 ： ループ文。返り値がTRUEの間、while文中の文が繰り返し実行される。

13行目 ： int型変数countをインクリメントする。

14行目 ： 条件分岐。countの値が5より大きくなったらbreak、while文を抜け出す。

16行目 ： printf文。「Enter a HexValue ===> \n」を出力する。

17行目 ： 標準関数fgetsで、キーボードから文字列を読み込む。

18行目 ： char型配列lineに格納されている文字列を16進数に変換し、変数cに代入。

20行目 ： printf文。変数countを0詰め2桁で、変数cを10進数に変換し右詰め3桁、
16進数に変換し右詰め2桁で出力。

21-24行目 ： if-else文。変数cが0x20以上かつ0x7e以下であるなら、cの値に対応する
ASCII文字を出力。そうでなければ、「-Not Printable character\n」を出力。

26-36行目 ： if-else if文。
変数cが0x20以上かつ0x2f以下であるなら、「====> Scope_A\n」を出力、
0x30以上かつ0x39以下であるなら、「====> Scope_B\n」を出力、
0x3a以上かつ0x40以下であるなら、「====> Scope_C\n」を出力、
0x41以上かつ0x5a以下であるなら、「====> Scope_D\n」を出力、
変数cがそれ以外の値であるなら、「====> Scope_E\n」を出力する。

＊ ASCIIコード(0x00～0x7f)の各範囲(Scope)を判断するプログラム

```
26      if      (0x30 <= c && c <= 0x39){
27        puts("====> figures\n");
28      }else if(0x41 <= c && c <= 0x5a){
29        puts("====> capital letter\n");
30      }else if(0x61 <= c && c <= 0x7a){
31        puts("====> small letter\n");
32      }else if(0x00 <= c && c <= 0x1f || c == 0x7f){
33        puts("====> Not Printable character\n");
34      }else{
35        puts("====> misc\n");
36      }
```

サンプルプログラムの26-36行を上のように変更する。

・0x30～0x39の範囲をfigures(数字)
・0x41～0x5aの範囲をcapital letter(英大文字)
・0x61～0x7aの範囲をsmall letter(英小文字)
・0x00～0x1f、0x7fの範囲をNot Printable character(非表示文字)
・0x20～0x2f、0x3a～0x40、0x5b～0x60、0x7b～0x7eの範囲をmisc(その他)

ASCIIコードを以上の5つの範囲に分け判断する。

実行結果

```
Enter a HexValue ===> 00
Colum=01:%d(  0)-%x( 0)-Not Printable character
====> Not Printable character

Enter a HexValue ===> 30
Colum=02:%d( 48)-%x(30)-%c(0)
====> figures

Enter a HexValue ===> 41
Colum=03:%d( 65)-%x(41)-%c(A)
====> capital letter

Enter a HexValue ===> 61
Colum=04:%d( 97)-%x(61)-%c(a)
====> small letter

Enter a HexValue ===> 5b
Colum=05:%d( 91)-%x(5b)-%c([)
====> misc
```

考　察

ASCIIコードの各範囲を判断するプログラムを作成し、非表示文字の範囲の判断が正しいかチェックしている時、次のような実行結果になった。

```
Enter a HexValue ===> 00
Colum=01:%d(  0)-%x( 0)-Not Printable character
====> misc

Enter a HexValue ===> 05
Colum=02:%d(  5)-%x( 5)-Not Printable character
====> misc

Enter a HexValue ===> 1e
Colum=03:%d( 30)-%x(1e)-Not Printable character
====> misc

Enter a HexValue ===> 1f
Colum=04:%d( 31)-%x(1f)-Not Printable character
====> misc

Enter a HexValue ===> 7f
Colum=05:%d(127)-%x(7f)-Not Printable character
====> misc
```

非表示文字0x00～0x1f、0x7fの範囲が、全てmisc(その他)の範囲として出力されている。
コンパイル時にエラーは出なかったので、各範囲を判断する26-36行目を見直してみると、非表示文字の範囲の条件が次のようになっていて、

```
if(0x00 <= c && c <= 0x1f && c == 0x7f)
```

条件式として成り立っていないことに気づいた。
条件式を成り立たせるため「c == 0x7f」は、&&(かつ)ではなく、||(または)にすることで、

```
if(0x00 <= c && c <= 0x1f || c == 0x7f)
```

非表示文字0x00～0x1f、0x7fの範囲を正しく判断できるプログラムになった。

2. sample#2.c のプログラムの動作を考察せよ。

＊ sample#2.c：while 文&for 文

```
 1  #include <stdio.h>
 2
 3  #define FALSE 0
 4  #define TRUE  !FALSE
 5
 6  int main(){
 7      int  count;
 8
 9      count = 0;
10      while(TRUE){
11        count++;
12        if(count > 5) break;
13        printf("While-Count=%2d\n",count);
14      }
15
16      for(count=1; count<=5; count++){
17        printf("for  -Count=%2d\n",count);
18      }
19
20      return(0);
21  }
```

出力結果

```
While-Count= 1
While-Count= 2
While-Count= 3
While-Count= 4
While-Count= 5
for  -Count= 1
for  -Count= 2
for  -Count= 3
for  -Count= 4
for  -Count= 5
```

考　察

while 文は、ループに入る前に変数 count を初期化をして、

・ループ文の中で変数 count をインクリメント
・条件分岐。変数 count が 5 より大きくなったら break、while 文から抜ける。
・printf 文。count の値を右詰め 2 桁で出力。

以上の 3 つの文を条件式 TRUE の間繰り返す。

for 文は、初期値を count = 1 に設定して、順次 count の値をインクリメントする。

・printf 文。count の値を右詰め 2 桁で出力。

以上の文を count の値が 5 になるまで繰り返す。

繰り返し処理ができる while 文と for 文は大きな違いがないと思われるが、
for 文の方は、文が少なくて済むということと、初期値、終値、増分の値を手軽に設定できることから、繰り返し処理では、for 文の方が使い易いと思う。
while 文は、do～while という使い方があるらしく、ループ処理の条件判断を後の方に持っていくことができ、条件に関係なく 1 度は処理させたい時に利用できるなどの特徴もあるので、処理に応じて使い分けることが大切だと思う。

3. sample#3.c を解析し、表示可能な文字による ASCII コード表を作成せよ。

＊ sample#3.c：文字コード（16進数&文字）出力

```
 1  #include <stdio.h>
 2
 3  int main(){
 4      int c;
 5
 6      for(c = 0x20; c<=0x40; c++){
 7        if((c % 4) == 0) printf("\n");
 8        printf("%%x(%x)-%%c(%c) | ",c,c);
 9      }
10      printf("\n");
11
12      return(0);
13  }
```

出力結果

```
%x(20)-%c( ) | %x(21)-%c(!) | %x(22)-%c(") | %x(23)-%c(#) |
%x(24)-%c($) | %x(25)-%c(%) | %x(26)-%c(&) | %x(27)-%c(') |
%x(28)-%c(() | %x(29)-%c()) | %x(2a)-%c(*) | %x(2b)-%c(+) |
%x(2c)-%c(,) | %x(2d)-%c(-) | %x(2e)-%c(.) | %x(2f)-%c(/) |
%x(30)-%c(0) | %x(31)-%c(1) | %x(32)-%c(2) | %x(33)-%c(3) |
%x(34)-%c(4) | %x(35)-%c(5) | %x(36)-%c(6) | %x(37)-%c(7) |
%x(38)-%c(8) | %x(39)-%c(9) | %x(3a)-%c(:) | %x(3b)-%c(;) |
%x(3c)-%c(<) | %x(3d)-%c(=) | %x(3e)-%c(>) | %x(3f)-%c(?) |
%x(40)-%c(@) |
```

解　析

4行目 ： int 型変数、c(16進数の値を格納)を定義。

6-9行目 ： 初期値をｃ = 0x20に設定、ｃが0x40以下の間処理を繰り返す。
順次変数ｃをインクリメントする。

7行目 ： 変数ｃを4で割った余りが0と等しかったら、改行する。
→ 4文字出力したら改行ということ。

8行目 ： printf 文。変数ｃを16進数に変換し出力。ｃの値に対応する ASCII 文字を出力。

10行目 ： 改行

12行目 ： return 文。main 関数に0を返す。

＊ 表示可能な文字による ASCII コード表

```
 1  #include<stdio.h>
 2
 3  int main(){
 4      int c;
 5
 6      for(c = 0x20; c <= 0x7e; c++){
 7         if((c % 4) == 0) printf("\n");
 8         printf("%%x(%x) %%c(%c) | ",c,c);
 9      }
10      printf("\n");
11
12      return(0);
13  }
```

サンプルプログラムの６行目、for 文の条件式を上のように変更する。
・表示可能な SP(0x20)から~(0x7e)までを出力。

出力結果

```
%x(20) %c( ) | %x(21) %c(!) | %x(22) %c(") | %x(23) %c(#) |
%x(24) %c($) | %x(25) %c(%) | %x(26) %c(&) | %x(27) %c(') |
%x(28) %c(() | %x(29) %c()) | %x(2a) %c(*) | %x(2b) %c(+) |
%x(2c) %c(,) | %x(2d) %c(-) | %x(2e) %c(.) | %x(2f) %c(/) |
%x(30) %c(0) | %x(31) %c(1) | %x(32) %c(2) | %x(33) %c(3) |
%x(34) %c(4) | %x(35) %c(5) | %x(36) %c(6) | %x(37) %c(7) |
%x(38) %c(8) | %x(39) %c(9) | %x(3a) %c(:) | %x(3b) %c(;) |
%x(3c) %c(<) | %x(3d) %c(=) | %x(3e) %c(>) | %x(3f) %c(?) |
%x(40) %c(@) | %x(41) %c(A) | %x(42) %c(B) | %x(43) %c(C) |
%x(44) %c(D) | %x(45) %c(E) | %x(46) %c(F) | %x(47) %c(G) |
%x(48) %c(H) | %x(49) %c(I) | %x(4a) %c(J) | %x(4b) %c(K) |
%x(4c) %c(L) | %x(4d) %c(M) | %x(4e) %c(N) | %x(4f) %c(O) |
%x(50) %c(P) | %x(51) %c(Q) | %x(52) %c(R) | %x(53) %c(S) |
%x(54) %c(T) | %x(55) %c(U) | %x(56) %c(V) | %x(57) %c(W) |
%x(58) %c(X) | %x(59) %c(Y) | %x(5a) %c(Z) | %x(5b) %c([) |
%x(5c) %c(\) | %x(5d) %c(]) | %x(5e) %c(^) | %x(5f) %c(_) |
%x(60) %c(`) | %x(61) %c(a) | %x(62) %c(b) | %x(63) %c(c) |
%x(64) %c(d) | %x(65) %c(e) | %x(66) %c(f) | %x(67) %c(g) |
%x(68) %c(h) | %x(69) %c(i) | %x(6a) %c(j) | %x(6b) %c(k) |
%x(6c) %c(l) | %x(6d) %c(m) | %x(6e) %c(n) | %x(6f) %c(o) |
%x(70) %c(p) | %x(71) %c(q) | %x(72) %c(r) | %x(73) %c(s) |
%x(74) %c(t) | %x(75) %c(u) | %x(76) %c(v) | %x(77) %c(w) |
%x(78) %c(x) | %x(79) %c(y) | %x(7a) %c(z) | %x(7b) %c({) |
%x(7c) %c(|) | %x(7d) %c(}) | %x(7e) %c(~) |
```

考　察

for 文を使うことで、プログラムがコンパクトにまとまり、わずか 13 行のプログラムで、表示可能な文字による ASCII コード表を出力することができた。
上記プログラムの 6-9 行目を、以下のように変更しても同様に出力できた。

```
c = 0x20;
while(TRUE){
  if(c == 0x7f) break;
  if((c % 4) == 0) printf("\n");
  printf("%%x(%2x) %%c(%c) | ",c,c);
  c++;
}
```

これより、for 文の処理を while 文で実現するのは比較的容易にできることがわかった。

4. 文字（文字列では無い）の演算について考察せよ。例）('a'-'A')?、('f'-'a')?

```
 1  #include<stdio.h>
 2
 3  int main(){
 4      int ans;
 5
 6      ans = 'a' - 'A';
 7      printf("\'a\' - \'A\' = %2d\n" ,ans);
 8
 9      ans = 'f' - 'a';
10      printf("\'f\' - \'a\' = %2d\n" ,ans);
11
12      return(0);
13  }
```

出力結果

```
'a' - 'A' = 32
'f' - 'a' =  5
```

考　察

ASCIIコード表より、a = 97 , A = 65 , f = 102
よって、

```
'a' - 'A' =  97 - 65 = 32
'f' - 'a' = 102 - 97 =  5
```

コンピュータ内部では、数値も文字も0と1で表現されているので、
文字どうしの演算ができるのだと思う。ここで、文字どうしの演算結果をまた文字として出力できるのではないかと考え、上のプログラムを以下のように変更し実行してみた。

```
ans = 'z' - '&';
printf("\'z\' - \'&\' = %2d\n" ,ans);
printf("\'z\' - \'&\' = %c\n",ans);
```

ASCIIコード表より、z = 122 , & = 38
よって、'z' - '&' = 122 - 38 = 84 なので、文字はTが出力されると考えられる。

```
'z' - '&' = 84
'z' - '&' = T
```

結果は予想通り、文字Tが出力された。
これにより、文字どうしの演算結果をまた文字として出力することが可能だということがわかる。文字を数値として扱うことで、任意の文字列を別の文字列に変換するなど様々な処理ができるといことがわかった。

考　察

今回は、ループ文の while 文や for 文、条件分岐の if 文などを用いて、ASCII コード表の出力、文字どうしの演算など、様々なプログラミングを行った。

for 文や while 文を使うことで、繰り返し処理するプログラムをコンパクトに、効率的にプログラミングできることがわかった。for 文は、初期値、終値、増分が設定でき、繰り返し処理が容易にできるので、とても使い易い関数だと感じた。

while 文は、for 文に比べ、変数を初期化したり、if 文で終了条件を定義したりと少々処理は多くなるが、do～while など後判定処理ができるという特徴があるので処理に応じて使い分ける必要があることが分かった。

if 文は、条件によって処理を行ったり、ループを抜けるなどの処理が行える、さらに、if-else if など重ねて条件分岐をすることで、より細かく条件分岐することができ、様々な出力結果を出すことができるということを知った。

また、コンピュータ内部では、文字も数値と同じように演算ができ、文字どうしの演算や、数値と文字の演算を組み合わせることで任意の文字列を別の文字列に変換できるのではないかと思う。

感　想

今回のプログラミングのレポートはとても時間がかかりました。特に、プログラムの解析は、何をやればいいのかよく分からなかったので、プログラム１行１行をどの様な処理をしているかというのをまとめてみました。
また、今回はレポートをまとめている際に「これもできるんじゃないか？」と色々、思い付いて実際にできたものを考察としてまとめてみました。
最後に、今回のレポートはまだ3回目なんですが、一番レポートらしいんじゃないかと思いますんで、どうかよろしくお願いします。

参考文献

Steve Oualline 著、谷口功 訳　『C実践プログラミング　第3版』　オーム社