１．sanple#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。

<サンプルプログラムソースコード>

```
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  }
41
42  /*
43  Enter a HexValue ===> 56
44  Colum=01:%d( 86)-%x(56)-%c(V)
45  ====> Scope_D
46
47  Enter a HexValue ===> 07
48  Colum=02:%d(  7)-%x( 7)-Not Printable character
49  ====> Scope_E
50
51  Enter a HexValue ===> ff
52  Colum=03:%d(255)-%x(ff)-Not Printable character
53  ====> Scope_E
54
55  Enter a HexValue ===> 7f
56  Colum=04:%d(127)-%x(7f)-Not Printable character
57  ====> Scope_E
58
59  Enter a HexValue ===> 7e
60  Colum=05:%d(126)-%x(7e)-%c(~)
61  ====> Scope_E
62  */
```

<解説>

このプログラムは与えられたサンプルをコピーしたもので、自分は一つも手を加えていません。また、実際に試し出力結果は４２～６２行までのコメント欄に例として載っているので省略しました。

これは大まかに、ユーザに１６進数の文字を入力させて、それに対応する数字を１０進数と１６進数で表示、さらにその数字に対応したASCII文字を表示し、入力した１６進数がどの範囲にあるかというものをA~Eの範囲で表示するプログラムです。

それでは、1行目から説明していきます。
3,4行目では、FALSEは0に、TRUEは!FALSEにするように設定されています。
その中で、7、8行目は変数の宣言であり、特に8行目は、要素数128個のchar型の配列lineを宣言しています。１０行目はループの回数を制御する変数countの、ループ直前の初期化を行っています。１２～３７行目では、while文を使用して繰り返し処理が行われています。１４行目で、whileが繰り返し処理するたびにcountにプラス1しています。
１４行目には、6回目のループでは、この行に続く処理が実行されないように、break文が書かれています。１６行目では、16進数の入力を要求するメッセージを出し、１７行目でlineの先頭から、sizeof(line)、すなわち、128(Byte)までは使用可能であることをfgets()関数に伝え、標準入

カストリームからlineに文字列を代入します。そして、１８行目でsscanf()関数を使用し、lineに入った文字列を%x(16進数)ととらえて、cに代入します。 ２０行目では、ループの回数と、cに入っている入力された値を10進数、16進数で表示し、２１~２４行目では、if文を使用して、表示可能な文字ならば文字として出力し、非表示文字なら、¬Not Printable characterという文字を出力しています。 ２６~３６行目では、文字をA~Eの5種類に分類して、その結果どのグループに所属しているかを表示しています。その処理を行う関数にはputs()関数を使用しています。 こういった一連の処理をwhileを用いて5回繰り返しています。 すべての処理が終わった３９行目で、正常終了を示す0をmain関数は返しています。

＜考察＞
while文とif文を使っていて、puts関数やfgets関数sscanf関数など見慣れない関数があってすこし戸惑いました。
注目すべき点は、表示可能な文字をすべて表示していない点と文字のグループ分けががあまり意味をなしていない点だと思います。これを参考にして、すべての文字を出力できてグループ分けができるプログラムを考えようと思います。

＜ソースコード＞

```
#include <stdio.h>

#define FALSE 0
#define TRUE  !FALSE

int main(){

  int  c;      /*16進数の入れ物*/
  int count;   /*ループの制御*/
  char line[128];

  count = 0;

  while(TRUE){
    count++;
    if(count > 5) break;

    printf("１６進数を入力してください ===> ");
    fgets(line, sizeof(line), stdin);
    sscanf(line, "%x", &c);

    printf("%d 回目の作業です。\n",count);
    printf("１６進数表記の%x は、１０進数で%d です。\n",c,c);
    if(0x21 <= c && c <= 0x7e)
```

```
    printf("対応する文字は(%c)です。\n",c);
   else
    printf("これは表示できません。\n");

   if     (0x00 <= c && c <= 0x1f){
    puts("====> 表示できません\n");
   }else if(0x20 <= c && c <= 0x2f){
    puts("====> 記号です\n");
   }else if(0x30 <= c && c <= 0x39){
    puts("====> 数字です\n");
   }else if(0x3a <= c && c <= 0x40){
    puts("====> 記号です\n");
   }else if(0x41 <= c && c <= 0x5a){
    puts("====> 大文字アルファベットです\n");
   }else if(0x5b <= c && c <= 0x60){
    puts("====> 記号です\n");
   }else if(0x61 <= c && c <= 0x7a){
    puts("====> 小文字アルファベットです\n");
   }else if(0x7b <= c && c <= 0x7e){
    puts("====> 記号です\n");
   }else{
    puts("====> 表示できません\n");
   }
  }

  return(0);
}

<出力結果>
[jun-no-macbook:~/prog1/kadai3] e095745% ./sample001
16進数を入力してください ===> 8
1回目の作業です。
16進数表記の8は、10進数で8です。
これは表示できません。
====> 表示できません

16進数を入力してください ===> 16
2回目の作業です。
16進数表記の16は、10進数で22です。
これは表示できません。
====> 表示できません

16進数を入力してください ===> 24
3回目の作業です。
```

１６進数表記の24は、１０進数で36です。
対応する文字は($)です。
====> 記号です

１６進数を入力してください ===> 32
4回目の作業です。
１６進数表記の32は、１０進数で50です。
対応する文字は(2)です。
====> 数字です

１６進数を入力してください ===> 40
5回目の作業です。
１６進数表記の40は、１０進数で64です。
対応する文字は(@)です。
====> 記号です

[jun-no-macbook:~/prog1/kadai3] e095745% ./sample001
６進数を入力してください ===> 48
1回目の作業です。
１６進数表記の48は、１０進数で72です。
対応する文字は(H)です。
====> 大文字アルファベットです

１６進数を入力してください ===> 56
2回目の作業です。
１６進数表記の56は、１０進数で86です。
対応する文字は(V)です。
====> 大文字アルファベットです

１６進数を入力してください ===> 64
3回目の作業です。
１６進数表記の64は、１０進数で100です。
対応する文字は(d)です。
====> 小文字アルファベットです

１６進数を入力してください ===> 72
4回目の作業です。
１６進数表記の72は、１０進数で114です。
対応する文字は(r)です。
====> 小文字アルファベットです

１６進数を入力してください ===> 80
5回目の作業です。
１６進数表記の80は、１０進数で128です。

これは表示できません。
====> 表示できません

<解説>
サンプルをヒントに改良を加えてみました。
改良点は
①日本語で出力した
②サンプルで"value"という変数があったが、必要ないと思い消去し、変数の使い道をコメントしている。
③SCII コード(0x00 ~ 0x7f)の範囲をすべて判別できるようにした

＜考察＞
出力結果は 8 の倍数を順番に入力していったものである。問題なくプログラムは起動しているのがわかります。

サンプルの"value"という変数が何に使われているのかわからなく、とても悩んだが消してしまっても問題はなかった。変数の宣言が大事だと思いコメントを記しめします。

2. sample#2.c のプログラムの動作を考察せよ

＜サンプルプログラムソースコード＞

```
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  }
```

```
22
23   /*
24   While-Count= 1
25   While-Count= 2
26   While-Count= 3
27   While-Count= 4
28   While-Count= 5
29   for -Count= 1
30   for -Count= 2
31   for -Count= 3
32   for -Count= 4
33   for -Count= 5
34   */
```

＜解説＞
２３～３４行目のコメントに出力結果が表示されています。

３、４行目でFALSEとTRUEを定義しています。
７行目でcountという変数を宣言し、９行目でそれに０を入れています。
１０～１４行目までがwhile文の範囲です。
while文の中でcountをプラス１し、countの現在の数字を出力しています。
この動作を繰り返し、５以上になった時breakが働き、反復が終了します。
１６～１８行目までがfor文の範囲です。for文の中でcountをまず初期値１に設定し、５以下の間反復し続けます。１７行目で現在の数字を表示して、また１をプラスする動作を繰り返し、countが６以上になった時反復が終了します。

＜考察＞
sample#1.cと使っている構文は一緒で、for文とwhile文のループを抜ける方法の違いをプログラムで説明しています。whileはcountが５以上になった時breakが発揮して反復が終了しているのに対して、forはcountが５以下であった時反復して６以上になった時、終了しているという違いがあります。

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。

＜ソースコード＞
```
1     int main(){
2       int c;
3
4       for(c = 0x20; c<=0x7e; c++){
5         if((c % 4) == 0) printf("\n");
6         printf("%%x(%x)-%%c(%c) |",c,c);
7       }
8       printf("\n");
```

```
 9
10      return(0);
11    }
```

＜出力結果＞

```
[jun-no-macbook:~/prog1/kadai3] e095745% ./sample003
%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(~) |
```

<解説>

sample#3.c を解析したところ、表示する文字の範囲が0x20 ~ 0x40とまだ足りないことに気づき、範囲を0x20 ~ 0x7eまで拡張しました。４行目はfor文で変数cに１づつプラスしていき、0x20から0x7f以上になった時反復を終了します。反復文は変数cを４で割り、あまりが０と同値になれば改行するということを表しています。
0x20はスペースであるため、表記されていないように見えますが記号としてあります。

＜考察＞

教科書のASCIIコード表を見ながら表示できる文字だけを出力できるようにしました。0x7fはデリートを表しているので表示文字ではなく出力していません。

出力結果から分かるように、半角で出力される文字と全角で出力される文字があるので、行がぼこぼこにになってしまい見ずらい表になってしまいました。これを改善する方法を考えたがわか

らなかったのが残念です。

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')?

<ソースコード>

```
#include <stdio.h>

int main(){
  int  kou, otu, hei, gai, mou, taku, tou;

  kou = 'a' + 'f';
  hei = 'a' + 'A';
  otu = '!' + 'd';
  gai = 'g' + '/';
  mou = 'k' + 'z';
  taku = '?' + ';';
  tou = '9' + '9';

  printf("10 進数→%d\n",kou,kou);printf("10 進数→%d\n",hei,hei);
  printf("10 進数→%d\n",otu,otu);printf("10 進数→%d\n",gai,gai);
  printf("10 進数→%d\n",mou,mou);printf("10 進数→%d\n",taku,taku);
  printf("10 進数→%d\n",tou,tou);

  return(0);
}
```

＜出力結果＞

```
[jun-no-macbook:~/prog1/kadai3] e095745% ./sample004
10 進数→199
10 進数→162
10 進数→133
10 進数→150
10 進数→229
10 進数→122
10 進数→114
```

＜解説＞

６つの変数を用意し、そのひとつひとつに文字の計算をいろいろしてみました。その計算結果を出力しています。

＜考察＞

変数に文字同士の足し算を入力しているのだが、数値として読み込んでいます。ASCIIコードに対応する数字が当てはまるだろうという予想は当たっていて、それほど不思議ではなかったです。一

度シングルクォーテーションでくくらず、ただa+aという風にやってコンパイルしたが無理でした。
注目すべき点は六番目の計算のtou = '9' + '9';である。９＋９＝１８ではなく対応する１０進数の５７で計算されたことです。

＜反省＞
For文やWhile文を使って自動に計算させたかったのだが、全く方法が浮かびませんでした。もっと使いこなせるようにならないといけないと思います。

＜感想＞
挫折しそうになるぐらい、とにかく難しかった。先輩のレポートを参考にしたのだが、自分と先輩のスキルが天と地のほどの差ほどあるというのが悔しかった。
すべてを使いこなすことができていない自分が腹立たしい。
次は納得できるレポートに仕上げたい。

＜参考文献＞
C実践プログラミング　第3版　Steve Oualline著
e065701 Akamine Kazuki先輩のページ　http://www.ie.u-ryukyu.ac.jp/~e065701/