1. 入力した正の整数を降順に並べ換えて出力するプログラムを作成せよ。
プログラムは個別にコンパイルし、make コマンドで実行すること。

> 入力データは 50 以下とし、以下の数が混在しているとする。
> 16 進数：先頭 1 文字が x または X（エックスの小文字か大文字）
> 8 進数：先頭 1 文字が 0（零）
> 10 進数：先頭 1 文字が 0（零）以外の数字

1.1 プログラム

1.1.1 ソースコード : cardinal.c

```
01  /*
02    Program    : cardinal.c
03    Comment    : 基数変換と整列処理
04  */
05
06  #include <stdio.h>
07  #include <string.h>
08  #define  MAX 256
09
10  void conv10(char **x, int *k, int n);
11  void select_sort(int x[], int m[], int n);
12  void print_num(char *x[], int m[], int n);
13  void msg();
14
15  int main(){
16    char *dt[50], num[10], buf[MAX], *p=buf;
17    int  n=0, len, i10[50], move[50];
18
19    puts("-------- Input");
20    while(gets(num) != NULL) {
21      len = strlen(num);
22      if(p > buf + MAX - len - 1 )  break;
23      strcpy(p, num);
24      dt[n] = p;
25      p += len + 1;
26      n++;
27    }
28    puts("-------- Result");
29    conv10(dt, i10, n);
30    select_sort(i10, move, n);
31    print_num(dt, move, n);
32    msg();
33
34    return(0);
35  }
```

1.1.2 ソースコード : conv10.c

```
01  /*
02    Program    : conv10.c
03  */
04
05  void conv10(char **x, int *k, int n){
06    while(n-- > 0){
07      switch(**x){
08        case '0' :
09          sscanf(*x + 1, "%o", k);
10          break;
11        case 'x' :
12        case 'X' :
13          sscanf(*x + 1, "%x", k);
14          break;
15        default :
16          sscanf(*x, "%u", k);
17          break;
18      }
19      x++;
20      k++;
21    }
22  }
```

1.1.3 ソースコード : select_sort.c

```
01  /*
02    Program    : select_sort.c
03  */
04
05  void select_sort(int x[], int m[], int n){
06    int  i, j, k, w;
07
08    for(i=0; i<n; i++)  m[i]=i;
09    for(i=0; i<n-1; i++){
10      k=i;
11      for(j=i+1; j<n; j++)
12        if(x[k] < x[j])  k=j;
13      w    = x[i];
14      x[i] = x[k];
15      x[k] = w;
16      w    = m[i];
17      m[i] = m[k];
18      m[k] = w;
19    }
20  }
```

1.1.4 ソースコード : print_num.c

```
01  /*
02    Program    : print_num.c
03  */
04
05  void print_num(char *x[], int m[], int n){
06    int  i;
07
08    for(i=0; i<n; i++){
09      puts( x[m[i]] );
10    }
11  }
```

1.1.5 ソースコード : msg.c

```
/*
  Program    : msg.c
*/
int msg(){
  printf("#### Message from C #### By Kouta,TOUME\n");
  return(0);
}
```

1.2 makefile

1.2.1 ソースコード : makefile

```
#
# cardinal.c _ makefile
#

cardinal : cardinal.o conv10.o select_sort.o print_num.o msg.o
	cc -o cardinal cardinal.o conv10.o select_sort.o print_num.o msg.o

cardinal.o : cardinal.c
	cc -c cardinal.c

conv10.o : conv10.c
	cc -c conv10.c

select_sort.o : select_sort.c
	cc -c select_sort.c

print_num.o : print_num.c
	cc -c print_num.c

msg.o : msg.c
	cc -c msg.c
```

1.2.2 実行結果 : makefile

```
new-host:Report8 kouta$ make -f makefile
cc -c cardinal.c
cc -c conv10.c
conv10.c: In function ‘conv10’:
conv10.c:9: warning: incompatible implicit declaration of built-in function
 ‘sscanf’
cc -c select_sort.c
cc -c print_num.c
cc -c msg.c
msg.c: In function ‘msg’:
msg.c:5: warning: incompatible implicit declaration of built-in function ‘printf’
cc -o cardinal cardinal.o conv10.o select_sort.o print_num.o msg.o
```

1.3 実行結果 : cardinal.c

```
new-host:Report8 kouta$ ./cardinal
-------- Input
warning: this program uses gets(), which is unsafe.
x21
021
X51
34
-------- Result
X51
34
x21
021
#### Message from C #### By Kouta,TOUME
```

1.4 考 察

1.4.1 main関数(1) : cardinal.c

20行目 : while文。条件文が gets(num) != NULL なので、”control + D”で終了。

21行目 : strlen関数で、numの文字列の長さを調べ、変数 len に格納。

23行目 : strcpy関数で、ポインタ変数p に配列num の文字列をコピーする。
p は buf[MAX]の先頭アドレスを持っているので、buf[MAX]に文字列をコピーしていることになる。

24行目 : 配列dt[n]に、pのアドレスを格納。

26行目 : 変数n をインクリメント。

main関数では、以下のような処理が行われている。

- 配列buf に、入力されたデータを格納。
- 配列dt には、入力されたデータそれぞれの先頭アドレスを格納。

よって、22行目のif文は、「配列buf にデータが入りきれないとき break する」という条件式になると考えられる。

- 配列buf の開いている領域は、buf + MAX - p
- 配列num に格納されている文字列の長さは、len + 1

以上のことより、buf + MAX - p - len -1 < 0 となり、if文の条件式は、

p > buf + MAX - len - 1

25行目の代入式について、配列buf に格納した文字列を上書きさせない為に格納した文字列分、p のアドレスも移動しなければいけない。

- 配列num に格納されている文字列の長さは、len + 1

よって、 p += len + 1 となる。

※ プログラム29～32行目の各関数への引数は後で解説する。

1.4.2 conv10関数 : conv10.c

6行目 : while文。n が 0 より小さくなるまで、処理を繰り返す。
これより、変数n は、入力された文字列の個数だと考えられる。

7行目 : switch文。**x の先頭の文字が、'0' or 'x' or 'X' or それ以外の場合に処理を行っている。

**x は入力された文字列を格納していると考えられる。

main関数で文字列を格納しているのは、配列buf であるが、ここでは1文字取り出したいので、

入力されたデータそれぞれの先頭アドレスを格納している、配列dt が **x となる。

sscanf関数について
- 書式 : sscanf(*str, format, *temp);　※ format は書式指定文字列。
- 意味 : 文字列*str を format の書式に変換し、配列temp に格納。

8行目 : **x が '0'のとき、つまり8進数のとき以下のような処理が行われる。
- 文字列を %o(8進数)に変換し、配列k に格納。

11-12行目 : **x が 'x', 'X'のとき、16進数とき以下のような処理を行う
- 文字列を %x(16進数)に変換し、配列k に格納。

15行目 : それ以外の場合、つまり10進数のときは、
- 文字列を %u(符号なし10進数)に変換し、配列k に格納。

8進数と16進数の先頭1文字は、場合分けの文字なので、1文字ずらす必要がある。
よって、*x +1 となる。
10進数の場合は、識別する文字がないので、*x となる。

while文で入力された文字列分、処理が繰り返されるので、以下の値を更新する必要がある。
- 参照する文字列の先頭アドレス*x
- 変換した文字列を格納する配列k

1つずらせば良いので、19, 20行目はそれぞれ、x++ , k++ となる。

1.4.3 select_sort関数 : select_sort.c

8行目 : 配列m に 0 から n(文字列の数)まで、順番よく数値を格納する。

9-12行目 : 先頭から順に値を確定していく、選択ソートである。
降順に並べるので、if文の条件式は、`x[k] < x[j]` となる。

13-18行目 : for文、if文の結果に応じて値を入れ替える。
配列x に対応させて、配列m も同様に入れ替える。

1.4.4 print_num関数 : print_num.c

8行目 : 0 から n(入力した文字列の数)まで、処理を繰り返す。

select_sort関数で、配列m に0 から n までの数値を順に格納し、配列x と対応させて、配列m もソートしている。

つまり、配列m は、ソートする前の配列x の添字を保持していることになる。

よって、9行目のputs関数で、`x[m[i]]` を出力となる。

1.4.5 msg関数 : msg.c

ソート終了後にメッセージを出力するための関数。

1.4.6 main関数(2) : cardinal.c

各関数への引数について。

（ｉ）conv10関数

char **x は、入力した文字列それぞれの先頭アドレスを格納している、配列dt 。

int *k は、変換後のデータを格納するため空の配列の配列i10 。

int n は、入力された文字列の数なので、変数n 。

（ⅱ）select_sort関数

int x[] は、ソートする値なので変換されたデータが格納されている、配列i10 。

int m[] は、ソート前の配列x の添字の役割をするので、空の配列の配列move 。

int n は、入力された文字列の数なので、変数n 。

（ⅲ）print_num関数

char *x[] は、入力されたデータが格納されている、配列dt 。

int m[] は、配列x の添字が格納されている配列move 。

int n は、入力された文字列の数なので、変数n 。

2. リスト構造プログラムの動作を考察しなさい。

2.1 ソースコード : list.c

```
01  /*
02    Program    : list.c
03    Comment    : リスト構造
04  */
05
06  #include <stdio.h>
07  #include <stdlib.h>
08
09  #define FALSE 0
10  #define TRUE  !FALSE
11
12  typedef struct Node{
13    int num;
14    struct Node *next_ptr;
15  }node;
16
17  node *start_ptr = NULL;
18
19  void ins(int idata){
20    node *p = start_ptr;
21
22    start_ptr = (node *)malloc(sizeof(node));
23    if (start_ptr == NULL) puts("Not enough memory!"), exit(0);
24
25    start_ptr->num = idata;
26    start_ptr->next_ptr = p;
27  }
28
29  int main(){
30    int idata;
31    node *p;
32
33    puts("Enter a sequence of integers:");
34    while(scanf("%d", &idata) == TRUE) ins(idata);
35
36    puts("In reverse order:");
37    for(p = start_ptr; p != NULL; p = p->next_ptr){
38      printf("Address=%p, Data=%3d\n", p, p->num);
39    }
40    puts("/end/");
41
42    system("PAUSE");
43    return(0);
44  }
```

2.2 実行結果 : list.c

```
new-host:Report8 kouta$ ./list
Enter a sequence of integers:
21
13
56
78
45
In reverse order:
Address=0x100160, Data= 45
Address=0x100150, Data= 78
Address=0x100140, Data= 56
Address=0x100130, Data= 13
Address=0x100120, Data= 21
/end/
sh: PAUSE: command not found
```

2.3 考　察 : list.c

12-15行目 : typedef を使うことで、nodeという構造体の型を宣言する。
自分自身と同じタグの構造体をポインタで宣言している。

17行目 : 先頭のポインタにNULLを設定する。

void型関数ins : 領域確保とデータの格納、リストの連結をしている。

20行目 : node型ポインタ変数p を宣言。start_ptrのアドレスを格納。

malloc関数について

- 書式 : (void *)malloc(size)
- 意味 : (void *)のためのメモリを size 分だけ確保する。

22行目 : (node *)のためのメモリを nodeの大きさ分だけ確保する。
そのメモリの先頭アドレスを start_ptr に格納。

23行目 : start_ptr が NULLならば、言い換えるとメモリが確保できなければ、
「Not enough memory!」という文を出力する。

25行目 : int型変数num に idata を格納。

26行目 : 今までの先頭ポインタを次のポインタp にする。

main関数 : 整数の入力、入力された値を逆に出力。

34行目 : 整数が入力されたらidata に格納し、関数int に渡す。

37行目 : 次のポインタが NULL になるまで処理を繰り返す。

- このプログラムを終了するには、「 control + D 」
- malloc関数を使うことで、メモリを無駄なく動的に確保している。
- 構造体へのポインタを参照するときは、「 ->（アロー演算子）」というのを使う。
- 配列などと違い、要素の追加・削除が比較的容易に行える

3. 感　想

今回のプログラミングのレポートは非常に時間がかかりました。

最初のソートと基数変換のプログラムの穴埋めは予想以上に難しくて、授業でメモしたものを見ながらやっと解けるという感じでした。
図など描きながら、何度もプログラムを見てるとだんだん意味が分かってきて、どんな処理をしているのかも分かった気がします。

リスト構造プログラムの考察は、非常に難しくて、普段はまとめながら理解できるのですが、今回は、何となくは理解できたのですが、当たっているか少し不安です。特に構造体をポインタとして使うところは未だによく理解できていないので、理解したいと思います。

4. 参考文献

- C実践プログラミング 第3版 谷口 功(訳)　望月 康司(監訳)　Steve Oualline(著)
- 初心者のためのポイント学習C言語　　http://www9.plala.or.jp/sgwr-t/