Lesson 28

Objectives: -

- Write a program to calculate permutation

Permutations

Function

This program displays all possible permutations of *n* data items.

Programming Issues

The algorithm used is fairly simple. For each item *i* in a set of *n* items:

1. Remove item *i* from the data set;
2. Find all permutations of the remaining *n - 1* items;
3. Insert item *i* back into the set in its original position

The recursive function `Perm()` shows this algorithm. Each permutation is printed as it is found.

The rest of the code is relatively lengthy, but merely supports the above algorithm by providing it with appropriate data structures. Two data structures are provided:

- A *data set*, an ordered list allowing two operations:
    1. Removing an item from a given position, and
    2. Inserting an item into a given position
- A *stack* to hold the current permutation, which like all stacks allows two operations:
    1. Pushing an item onto the top of the stack, and
    2. Popping an item off the top of the stack

In order to understand the algorithm used to find the permutations, it is not necessary to delve into the details of these functions or data structures any more than it is necessary to understand exactly how a `printf()` call produces output on your screen. The `main()` and `Perm()` functions describe the algorithm completely. To demonstrate this, two versions of the program are supplied, one using linked lists, and one using arrays. Aside from the files `data_####.c` and `stack_####.c` the two versions are identical.

Functions to:

- Initialize and free memory used by the data set,
- Print out the contents of the stack, and
- Retrieve the number of data items from the command line

are also provided.

Click here for some performance statistics.

Usage

Pass the number of data items to the command line as the single argument. An example session is shown below:

```
[paul@hermes perm]$ ./perm 3
Permutations of 3 items:

ABC
ACB
BAC
BCA
CAB
CBA

6 permutations in all.
```

perm.c

```
/*
  PERM.C
  ======
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net
```

```
  Prints all the possible permutations of 'n'
items.
  'n' must be supplied as the sole command line
argument.
*/


#include <stdio.h>
#include <stdlib.h>

#include "data.h"
#include "stack.h"
#include "cmdline.h"


static void Perm(unsigned n);

static int count = 0;


int main(int argc, char *argv[]) {
    unsigned n = ParseCmdLine(argc, argv);

    InitializeData(n);

    printf("Permutations of %u items:\n\n", n);
    Perm(n);
    printf("\n%d permutations in all.\n", count);

    FreeData();

    return EXIT_SUCCESS;
}

void Perm(unsigned n) {
    unsigned item;

    if ( n == 0 ) {
        PrintStack();
        ++count;
        return;
    }

    for ( item = 0; item < n; ++item ) {
        Push(RemoveItem(item));
        Perm(n-1);
        InsertItem(Pop(), item);
    }
}
```

---

data.h

```
/*
```

```
  DATA.H
  ======
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  Interface to functions for dealing with data
*/


#ifndef PG_DATA_H
#define PG_DATA_H

void InitializeData(unsigned n);
void FreeData(void);
char RemoveItem(unsigned pos);
void InsertItem(char n, unsigned pos);

#endif  /* PG_DATA_H */
```

---

data_array.c

```
/*
  DATA_ARRAY.C
  ============
  (c) Copright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  Functions for manipulating the data using
arrays
*/


#include <stdio.h>
#include <stdlib.h>

#include "data.h"


static char data[27] =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
static unsigned size;


/*  Initialize memory for data  */

void InitializeData(unsigned n) {
    size = n;
}


/*  Free memory used for data  */

void FreeData(void) {
    return;
```

```
}


/*  Remove data item at position 'pos'  */

char RemoveItem(unsigned pos) {
    char d = data[pos];

    if ( pos >= size-- ) {
        fprintf(stderr, "Data position %u is out
of bounds.\n", pos);
        exit(EXIT_FAILURE);
    }

    while ( pos < size ) {
        data[pos] = data[pos+1];
        ++pos;
    }

    return d;
}


/*  Insert data item 'n' at position 'pos'  */

void InsertItem(char n, unsigned pos) {
    int i = (signed) size;

    if ( pos > size++ ) {
        fprintf(stderr, "Data position %u is out
of bounds.\n", pos);
        exit(EXIT_FAILURE);
    }

    while ( --i >= (signed) pos )
        data[i+1] = data[i];

    data[pos] = n;
}
```

---

stack.h

```
/*
  STACK.H
  =======
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  Interface to stack operations
*/


#ifndef PG_STACK_H
#define PG_STACK_H
```

```
void InitializeStack(void);
void Push(char n);
char Pop(void);
void PrintStack(void);

#endif  /* PG_STACK_H */
```

---

stack_array.c

```
/*
  STACK_ARRAY.C
  =============
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  Stack operations using arrays
*/


#include <stdio.h>
#include <stdlib.h>

#include "stack.h"


#define MAX_STACK_SIZE 26

static char stack[MAX_STACK_SIZE];
static int top = -1;


/*  Push item 'n' onto the stack  */

void Push(char n) {
    if ( ++top == MAX_STACK_SIZE ) {
        fprintf(stderr, "Stack full!\n");
        exit(EXIT_FAILURE);
    }

    stack[top] = n;
}


/*  Pop top item from the stack  */

char Pop(void) {
    if ( top < 0 ) {
        fprintf(stderr, "Stack empty!\n");
        exit(EXIT_FAILURE);
    }

    return stack[top--];
}
```

```
/*  Output contents of stack  */

void PrintStack(void) {
    int i = 0;

    while ( i <= top )
        putchar(stack[i++]);

    putchar('\n');
}
```

---

cmdline.h

```
/*
  CMDLINE.H
  =========
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  Interface to command line parsing function
*/


#ifndef PG_CMDLINE_H
#define PG_CMDLINE_H

unsigned ParseCmdLine(int argc, char *argv[]);

#endif /* PG_CMDLINE_H */
```

---

cmdline.c

```
/*
  CMDLINE.C
  =========
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  Function to parse command line and return
integral argument
*/


#include <stdio.h>
#include <stdlib.h>

#include "cmdline.h"
```

```
unsigned ParseCmdLine(int argc, char *argv[]) {
    unsigned n;
    char * endptr;

    if ( argc < 2 ) {
        fprintf(stderr, "You must supply an
argument\n");
        exit(EXIT_FAILURE);
    }
    else if ( argc > 2 ) {
        fprintf(stderr, "You must only supply one
argument\n");
        exit(EXIT_FAILURE);
    }

    n = (unsigned) strtoul(argv[1], &endptr, 0);

    if ( *endptr ) {
        fprintf(stderr, "You must supply a whole
number as an argument\n");
        exit(EXIT_FAILURE);
    }
    else if ( n > 26 ) {
        fprintf(stderr, "You must specify a
number less than 27\n");
        exit(EXIT_FAILURE);
    }
    else if ( n < 1 ) {
        fprintf(stderr, "You must specify a
number greater than 0\n");
        exit(EXIT_FAILURE);
    }

    return n;
}
```

Questions: -

1. Write a program to calculate permutation