Computer programming

"He who loves practice without theory is like the sailor who boards ship without a ruder and compass and never knows where he may cast."

Leonardo da Vinci

Outline

- File handling
  - High level I/O
    - fopen
    - fclose
    - fread
    - fwrite
    - fsetpos, fgetpos, ftell, fseek
- Applications
  - Combinatorial generation: generating subsets
    - Cross product (Cartesian product)
    - Combinations
    - Permutations
    - Arrangements
    - Power set

High level I/O. Files and Streams

- C views each file as a sequence of bytes
  - File ends with the *end-of-file marker*
  - Or, file ends at a specified byte
- Stream created when a file is opened
  - Provide communication channel between files and programs
  - Opening a file returns a pointer to a **`FILE`** structure
    - Example file pointers:
    - **`stdin`** - standard input (keyboard)
    - **`stdout`** - standard output (screen)
    - **`stderr`** - standard error (screen)
- **`FILE`** structure
  - File descriptor - Index into operating system array called the open file table
  - File Control Block (FCB) - Found in every array element, system uses it to administer the file

Files and Streams

- To process a file:
  - Open file
  - Do work (read, write)
  - Close file
- Opening a file = request
  - Must check that the request was accepted. i.e. a non-NULL pointer was returned

```
myPtr = fopen("myFile.dat", "r");
If (myPtr == NULL) {
// cannot continue, file not opened
}
```

Files and Streams

- Read/Write functions in standard library
  - `fgetc` - reads one character from a file
    - Takes a `FILE` pointer as an argument
    - `fgetc( stdin )` equivalent to `getchar()`
  - `fputc` - writes one character to a file
    - Takes a `FILE` pointer and a character to write as an argument
    - `fputc( 'a', stdout )` equivalent to `putchar( 'a' )`
  - `fgets` - read a line from a file
  - `fputs` - write a line to a file
  - `fscanf / fprintf` - file processing equivalents of `scanf` and `printf`

Creating a Sequential Access File

- C imposes no file structure
  - No notion of records in a file
  - Programmer must provide file structure
- Creating/using a file: **`fopen`**
  - **`FILE *myPtr;`** - creates a **`FILE`** pointer
  - **`myPtr = fopen("myFile.dat", openmode);`**
    - Function **`fopen`** returns a **`FILE`** pointer to file specified
    - Takes two arguments - file to open and file open mode
    - If file not opened, **`NULL`** returned

Creating a Sequential Access File

- **`feof(FILE pointer)`** - returns **`true`** if end-of-file indicator (no more data to process) is set for the specified file

- **`fclose(FILE pointer)`** - closes specified file
  - Performed automatically when program ends
  - Good practice to close files explicitly

- Details
  - Each file must have an unique name and will have a different pointer
    - All file processing must refer to the file using the pointer

- `filePointer = fopen(filePath, openmode)`
- Each mode can have a 'b' (for binary, e.g. ab, wb, rb+) after mode letter
- `openmode` given as a string ("r", "w", "rb", "wb", "r+" etc.)

Mode	Description
r	Open a file for reading.
w	Create a file for writing. If the file already exists, discard the current contents.
a	Append; open or create a file for writing at end of file.
r+	Open a file for update (reading and writing).
w+	Create a file for update. If the file already exists, discard the current contents.
a+	Append; open or create a file for update; writing is done at the end of the file.

Attaching a file to an open stream

- **`freopen(const char *filename, const char *mode, FILE *stream)`**
  - Most common use: associate a file with one of the standard streams
  - Example: cause program begin writing to **`foo.txt`**

```
if (freopen("foo.txt", "w", stdout) == NULL)
{
  // error foo.txt cannot be opened
}
```

  - Effect: close any other file previously associated to **`stdout`** ; then open **`foo.txt`** and associate it with **`stdout`**.

Reading Data from a Sequential Access File

- Reading a sequential access file
  - Create a **`FILE`** pointer, link it to the file to read

```
myPtr = fopen( "myFile.dat", "r" );
```

  - Use **`fscanf`** to read from the file
    - Like **`scanf`**, except first argument is a **`FILE`** pointer

```
fscanf( myPtr, "%d%s%f", &myInt, &myString,
  &myFloat );
```

  - Data read from beginning to end

Reading Data from a Sequential Access File

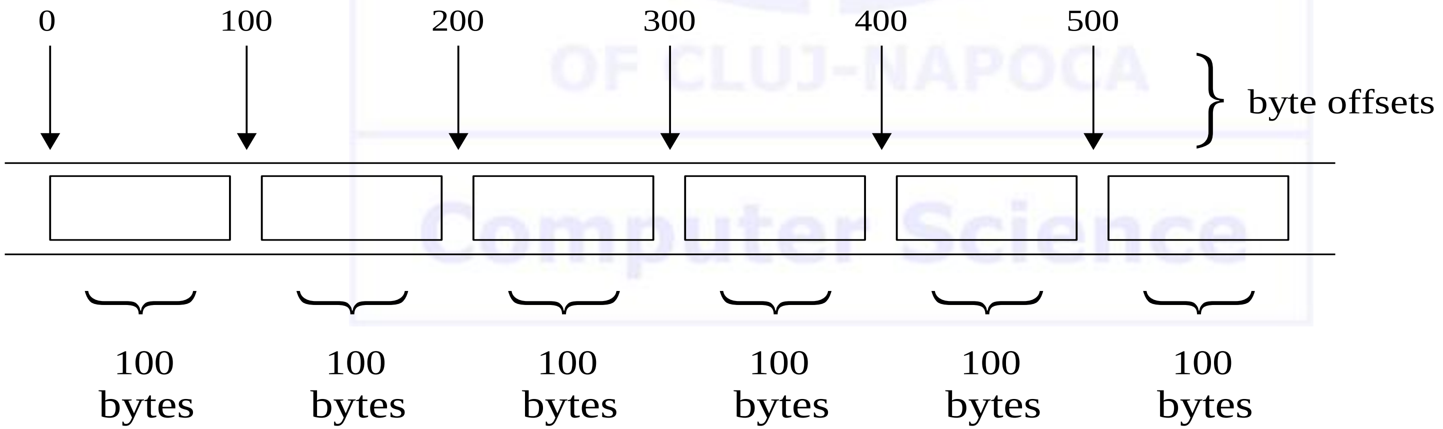
- File position pointer - indicates number of next byte to be read/written
  - Not really a pointer, but an integer value (specifies byte location)
  - Also called byte offset
- **`rewind(myPtr)`** - repositions file position pointer to beginning of the file (byte 0)
- Cannot be modified without the risk of destroying other data

Random Access Files

- Random access files
  - Access individual records without searching through other records
  - Instant access to records in a file
  - Data can be inserted without destroying other data
  - Data previously stored can be updated or deleted without overwriting.

Creating a Random Access File

- Implemented using *fixed length* records
  - Sequential files do not have fixed length records
- Data unformatted (stored as "raw bytes") in random access files
  - All data of the same type (**`int`**s, for example) use the same memory
  - All records of the same type have a fixed length
  - Data not human readable



Creating a Random Access File

- Unformatted I/O functions
  - `fwrite` - Transfer bytes from a location in memory to a file
  - `fread` - Transfer bytes from a file to a location in memory
  - `fwrite( &number, sizeof( int ), 1, myPtr );`
    - `&number` - Location to transfer bytes from
    - `sizeof( int )` - Number of bytes to transfer
    - `1` - For arrays, number of elements to transfer
      - In this case, "one element" of an array is being transferred
    - `myPtr` - File to transfer to or from
    - `fread` similar

Writing Data Randomly to a Random Access File

- Writing data : `fwrite`

  ```
  size_t fwrite(const void *ptr, size_t size, size_t
    nelem, FILE *stream);
  ```

  - `ptr` = pointer to memory area where info to write is stored
  - `size` = size in bytes of one element
  - `nelem` = number of elements to write
- E.g. writing `struct`s

  ```
  fwrite( &myObject, sizeof (struct myStruct), 1,
    myPtr );
  ```

  - `sizeof` - Returns size in bytes of object in parentheses
- To write several array elements
  - Pointer to array as first argument
  - Number of elements to write as third argument

Reading Data Sequentially from a Random Access File. Removing a file

- Reading data: `fread`

```
size_t fread(void *ptr, size_t size, size_t nelem, FILE *stream);
```

  - `ptr` = pointer to memory area where read info will be stored
  - `size` = size in bytes of one element
  - `nelem` = number of elements to write
  - Example:

```
fread( &client, sizeof (struct clientData), 1, myPtr );
```

  - Can read several fixed-size array elements
    - Provide pointer to array
    - Indicate number of elements to read
  - To read multiple elements, specify in third argument
- Removing a file

```
int unlink(const char *path_to_file)
```

  - returns 0 if successful, and -1 on error

File position

```
/* return file position indicator */
long ftell(FILE *stream);
int fgetpos(FILE *stream, fpos_t *pos);
/* set file position indicator to zero */
void rewind(FILE *stream);
/* set file position indicator */
int fseek(FILE *stream, long offset, int ptrname);
int fsetpos(FILE *stream, const fpos_t *pos);
```

- `ftell` returns the current value (measured in characters) of the file position indicator if stream refers to a binary file.
  - For a text file, a 'magic' number is returned, which may only be used on a subsequent call to `fseek` to reposition to the current file position indicator.
  - On failure, `-1L` is returned and `errno` is set.
- `rewind` sets the current file position indicator to the start of the file indicated by stream. The file's error indicator is reset by a call of rewind. No value is returned.

File position

- `int fseek(FILE *stream, long offset, int ptrname)`
- File position indicator for stream set to an arbitrary value (for binary files), or for text files, only to a position obtained from `ftell`, as follows:
  - For both functions, on success, zero is returned; on failure, non-zero is returned and `errno` is set.
  - General case: the file position indicator is set to offset bytes (characters) from a point in the file determined by the value of ptrname. Offset may be negative.

File position

- The values of `ptrname` may be `SEEK_SET`, `SEEK_CUR`, and `SEEK_END`. The latter is not necessarily guaranteed to work properly on binary streams.
- For text files, offset must either be zero or a value returned from a previous call to ftell for the same stream, and the value of `ptrname` must be `SEEK_SET`.
- `fseek` clears the end of file indicator for the given stream and erases the memory of any `ungetc`. It works for both input and output.

File position

- For `ftell` and `fseek` it must be possible to encode the value of the file position indicator into a `long`.
  - This may not work for very long files
  - `fgetpos` and `fsetpos` have been specified in a way that removes the problem.
- `fgetpos` stores the current file position indicator for stream in the object pointed to by pos.
  - The value stored is only used to return to the specified position for the same stream using `fsetpos`.
- `fsetpos` works as described above, also clearing the stream's end-of-file indicator and forgetting the effects of any `ungetc` operations.

Example: text file

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
   char ch, s[100], filename[]="textfile.txt";
   int i;
   FILE *fp;
   /* create file */
   fp = fopen(filename, "w");
   printf("\nInput lines of text for file. End with Ctrl/Z\n");
   while ((ch = getc(stdin)) != EOF) putc(ch, fp);
   fclose(fp);
   /* add text to file */
   fp = fopen(filename, "r+");
   fseek(fp, 0l, SEEK_END);
   printf("\nInput lines text to add to file. End with Ctrl/Z\n");
   while (fgets(s, sizeof(s), stdin) != NULL) fputs(s, fp);
   fclose(fp);
   /* display contents */
   printf("\nLines of the file (numbered):\n");
   i=1;
   fp = fopen(filename, "r");
   while (fgets(s, sizeof(s), fp) != NULL) printf("%d: %s", i++, s);
   fclose(fp);

  return 0;
}
```

Files

- Text files
  - Characters are (usually) human readable
  - Are divided into lines
  - May contain a special “End-of-file” marker
  - Examples: source files, header files
- Binary files
  - None of the above characteristics
  - Examples: executables, object files, databases, etc.

Error Handling

- System calls set a *global* integer called **`errno`** on error:
  - **`extern int errno;   /* defined in errno.h */`**
- The constants that errno may be set to are defined in **`<errno.h>`**. For example:
  - **`EPERM`** operation not permitted
  - **`ENOENT`** no such file or directory (not there)
  - **`EIO`** I/O error
  - **`EEXIST`** file already exists
  - **`ENODEV`** no such device exists
  - **`EINVAL`** invalid argument passed

```
#include <stdio.h>
void perror(const char * s);
```

- E.G. if **`errno==EINVAL`**, then **`perror("EINVAL: ");`** prints:

```
EINVAL: : Invalid argument
```

File status

- `int stat(const char * pathname, struct stat *buf);`
- The `stat()` system call returns a structure (into a buffer you pass in) representing all the stat values for a given filename. This information includes:
  - the file's mode (permissions)
  - inode number (in Unix)
  - number of hard links
  - user id of owner of file
  - group id of owner of file
  - file size
  - last access, modification, change times
  - see header file `stat.h` and `sys/types.h` (`S_IFMT`, `S_IFCHR`, etc.)

File status example

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <time.h>
int main(int argc, char *argv[])
{
    char *fname="main.c";
    struct stat statBuf;

    if (0==stat(fname, &statBuf))
    {
      printf("\nFile: %s", fname);
      if (S_ISDIR (statBuf.st_mode))
         puts("\n\tdirectory");
      if (S_ISCHR(statBuf.st_mode))
         puts("\n\tcharacter special file");
      if (S_ISBLK(statBuf.st_mode))
         puts("\n\tblock special file");
      if (S_ISREG(statBuf.st_mode))
         puts("\n\tregular file");
      if (S_ISFIFO(statBuf.st_mode))
         puts("\n\tFIFO special file, or a pipe");
      printf("\tinode=%u\n\tdevice=%c
:", statBuf.st_ino,
statBuf.st_dev+'A');

      printf("\n\tlinks=%d\n\tuid=%d\
n\tgid=%d\n\tsize=%ld",
          statBuf.st_nlink,
statBuf.st_uid, statBuf.st_gid,
statBuf.st_size);
      printf("\n\taccess
time=%s\tcontents mod
time=%s\tattrib mod time=%s",
          ctime(&statBuf.st_atime),
ctime(&statBuf.st_mtime),
          ctime(&statBuf.st_ctime));
    }
    else perror(0);
  system("PAUSE");
  return 0;
}
```

File status example

- Program output example:

```
File: main.c
        regular file
        inode=0
        device=D:
        links=1
        uid=0
        gid=0
        size=1007
        access time=Thu Dec 21 08:47:51 2005
        contents mod time=Thu Dec 21 08:47:51 2005
        attrib mod time=Thu Dec 21 08:47:51 2005
```

Algorithm examples

- Cross product
- Combinations
- Permutations
- Power set

Cross product

- Consider n sets of positive integers: $A_1, A_2, \ldots, A_n$.
  - Set A_i , for $i=1, 2, \ldots, n$ has n_i elements
  - The cross product (Cartesian product, set direct product, product set) is required. i.e.

$$A_1 \times A_2 \times \ldots \times A_n = \prod_{i=1}^{n} A_i$$

  - having $\prod_{i=1}^{n} n_i$ elements which will be generated in a vector, $p=[\, p_1\, p_2 \ldots\, p_n]$

Cross product algorithm

- Set every element of vector p to 1.
  - This is the first element of the cross product
- Find next element as follows:
  - Find the highest index i for which $p_i < n_i$.
    If such an index cannot be found then all elements have been generated. Stop
  - Find next element of the cross product as
    $\{p_1, p_2, p_{i-1}, p_{i+1}, 1, 1, \ldots, 1\}$

Cross product implementation. Non recursive

```
#include <stdio.h>
#define MAXN 10
void listProduct(int n, int
   prodNb, int p[])
{
   int i;
   printf("\n%3d ", prodNb);
   for (i = 1; i <=n; i++)
   printf(" %2d", p[i]);
   if ( prodNb % 20 == 0 )
   getch();
}
void crossProdNonRec(int n,
   int nElem[])
/*   n = number of sets;
   nElem = vector with number
   of elements per set */
{
   int i, prodNb, p[MAXN];
   prodNb = 1;
   for (i = 1; i <= n; i++) p[i] = 1;
   listProduct(n, prodNb, p);
   i = n;
   while ( i > 0 )
   {
      p[i]++; // find highest such as p[i] > nElem[i]
      if (p[i] > nElem[i])
      {
         p[i] = 1;
         i--;
      }
      else
      {
         prodNb++;
         listProduct(n, prodNb, p);
         i = n;
      }
   }
}
```

Cross product implementation. Non recursive

```
int main(int argc, char *argv[])
{
  int i, n, nElem[MAXN];
   printf("\nNumber of sets [<%d]=", MAXN); scanf("%d", &n);
   for (i = 1; i <= n; i++)
   {
       printf("Number of elements in set %d=", i);
       scanf("%d", &nElem[i]);
   }
   printf("\nThe members of the cross product");
   printf("\nNo.   Elements");
   crossProdNonRec(n, nElem);
  getchar();
  return 0;
}
```

Cross product implementation. Recursive

```
#include <stdio.h>
#include <conio.h>
#define MAXN 10
int prodNb, p[MAXN],
   nElem[MAXN];
void listProduct(int n)
{
   int i;

   printf("\n%3d ", prodNb);
   for (i = 1; i <=n; i++)
   printf(" %2d", p[i]);
   if ( prodNb % 20 == 0 )
   getch();
}
```

```
void crossProdRec(int n, int i)
/* n = number of sets; nElem =
   vector with number of elements
   per set */
{
   int j;

   for (j = 1; j <=nElem[i]; j++)
   {
     p[i] = j;
     if (i< n) crossProdRec(n, i+1);
     else
     {
       prodNb++;
       listProduct(n);
     }
   }
}
```

Cross product implementation. Recursive

```
int main(int argc, char *argv[])
{
  int i, n;
   printf("\nNumber of sets [<%d]=", MAXN); scanf("%d", &n);
   for (i = 1; i <= n; i++)
   {
       printf("Number of elements in set %d=", i);
       scanf("%d", &nElem[i]);
   }
   printf("\nThe members of the cross product");
   printf("\nNo.   Elements");
   crossProdRec(n, 1);
  getch();
  return 0;
}
```

Combinations

- Let P be a set of n elements
- All ways of picking k unordered elements of the n elements = generating all subsets with $k \leq n$ elements of P such as any two subsets are distinct
  - The number of subsets is the *binomial coefficient* or *choice number* and read "*n* choose *k*"

Combinations algorithm

- First subset is $p=\{1, 2, ..., k\}$
- Given a subset, its successor is found as follows:
  - Going from *k* down to 1 find index i which satisfies the relationships

$$
\begin{aligned}
p_i &< n-k+i \\
p_{i+1} &= n-k+i+1 \\
&\ldots \\
p_{k-1} &= n-1 \\
p_k &= n
\end{aligned}
$$

  - Successor set is:

$$\{p_1, p_2, ..., p_i+1, p_i+2, ..., p_i+n-k+1\}$$

  - The last subset is:

$$\{n-k+1, n-k+2, ..., n-1, n\}$$

Combinations algorithm implementation. Non recursive

```
void combinNonRec(int n, int k)
{
    int p[MAXN];
    int i, j, combinNb;
    for (i=1; i <=k; i++) p[i]=i; /* first combination */
    listCombin(k, combinNb, p);
    i = k;
    while (i > 0) /* generate the next combinations */
    {
      p[i]++; // find index satisfying relation set
      if (p[i] > n - k + i) i--;
      else
      {
        for (j = i + 1; ; j <= k; j++) p[j]=p[j-1] + 1;
        combinNb++;
        listCombin(k, combinNb, p);
        i = k;
      }
    }
}
```

Combinations algorithm implementation. Recursive

```
void combinRec(int n, int k, int i)
{
    int j;
    for (j = p[i-1]+1; j <= n-k+i; j++)
    {
      p[i] = j;
      if (i < k) combinRec(n, k, i+1);
      else
      {
        combinNb++;
        listCombin(k, combinNb, p);
      }
    }
}
```

- Notes
  - Array `p`, and `combinNb` must be global
  - Invocation is: `combinRec(n, k, 1)`

Generating Permutations

- For instance 35241 is the permutation that maps 1 to 3, 2 to 5, 3 to 2, 4 to 4, and 5 to 1.
- We know that there are 7! permutations on {1,2,3,4,5,6,7}.
  - Suppose we want to list them all. Is there an efficient way to do so? It turns out to be fairly simple to list them lexicographically.
  - The only hard question is, given one permutation, how do we find the next one?
- The lexicographically first permutation is

$$1\ 2\ \ldots\ n$$

- and the last is

$$n\ n\ \text{-}1\ \ldots\ 1$$

Generating Permutations

- It is intuitively reasonable that if the final digits of a permutation are in descending order, then no rearrangement will make them larger.
  - For instance in 125*7643* we cannot produce a larger number by rearranging the 7643.
  - Instead we must increase the next most significant digit (the 5) by the next larger digit in 7643 (the 6).
  - Then the remaining digits (the 5 and the 743) must be arranged to form the smallest number possible.
  - Thus the next permutation in lexicographic order is 1263457.

Generating permutations in lexicographical order

a. First permutation is $p = \{1, 2, \ldots n\}$

b. Given vector $p=[p_1\ p_2\ \ldots\ p_n]$ the next permutation is found as follows:

1. Look from n down to 1 for the highest valued index which satisfies the relationships:

$$p_i < p_{i+1}$$
$$p_{i+1} > p_{i+2} > \ldots > p_n$$

2. Find the maximum element, $p_k > p_i$ of $p_{i+1}, p_{i+2}, \ldots, p_n$
3. Swap p_k with p_i
4. Revert $p_{i+1}, p_{i+2}, \ldots, p_n$ by swapping p_{i+1} and p_n, p_{i+2} and p_{n-1}, a.s.o.

Permutations implementation. Non recursive

```
void permNonRec(int n) {
    int p[MAXN];
    int i, k, permNb = 0;

    /* first permutation, step a */
    for (i = 1; i <= n; i++) p[i] = i;
    listPerm(n, ++permNb, p);
    do /* generate the next permutations */
    {
        i = n - 1;
        while (p[i] > p[i+1] && i > 0) i--; /*
  step b1 */
        if (i > 0) {
            for (k = n; p[i] > p[k]; k--); /*
  step b2 */
            swap( &p[i], &p[k] );  /* step b3 */
            revert(p,  i, n, (n - i ) / 2);
            listPerm(n, ++permNb, p);
        }
    } while (i > 0);
}
```

```
void swap(int *i, int *j)
{
  int temp;
  temp = *i;
  *i = *j;
  *j = temp;
}

void revert(int p[], int i, int n,
  int k)
{
  int j;
  for (j = 1; j <= k; j++)
    swap(p[i+j], p[n+1-j]);
}
```

A note on swapping

- Swapping integers in place

```
void swap(int *i, int *j)
{
   *i += *j; // i == i + j
   *j = *i - *j;// j == i + j - j == i
   *i -= *j; // i == i + j - i == j
}
void swap(int *i, int *j)
{
   *i ^= j; // a == a^b;
   *j ^= *i;// b == b^(a^b)
   *i ^= *j;// a == (a^b)^(b^(a^b))
}
```

Second version based on proof on the nearby table, where a and b are bit positions

final b

final a

a	b	a^b	b^(a^b)	(a^b)^(b^(a^b))
0	0	0	0	0
0	1	1	0	1
1	0	1	1	0
1	1	0	1	1

Permutations implementation. Recursive

```
void permRec(int nb)
{
    int i, j;

    if ( nb == 1 )
    {
      permNb++;
      listperm();
    }
    else
    {
      permRec(nb - 1);
      for (i = 1; i <= nb - 1; i++)
      {
        swap( p[i], p[nb] );
        permRec(nb - 1);
        swap( p[i], p[nb] );
      }
    }
}
```

- Notes
  - Generates recursively permutations of elements p_1 p_2 ... $p_{n\text{-}1}$ with p_n in position n
  - Then swaps p_i with p_n for i =1..n-1, and generates all permutations
  - Array `p, n,` and `permNb` must be globals
  - First permutation must be initialized separately
  - Invocation: `permRec(n)`
  - Order is not loxicographic

Generating *k* permutations of *n*

- For set $p=\{1, 2, ..., n\}$ and $k \leq n$, positive
  - generate all ***k***-subsets such any two subsets must differ either in the composing elements or in their order
- Algorithm idea:
  - generate all combinations of n elements taken k, and, for each combination
    - generate the $k!$ permutations

Generating *k* permutations of *n* implementation

```
void arrange(int n, int m, int i) {
    int j, k, r;
    for (j = p[i-1] + 1; j <= n - m + i; j++) // recursively generate combinations
    {
        p[i] = j;
        if (i < m) arrange(n, m, i + 1);
        else
        {
            arrNb++;
            listArrang(m, p);

            for (k = 1; k <= m; k++) v[k] = p[k]; // save combination
            do   // nonrecursively permute combination
            {
                k = m - 1;
                while (v[k] > v[k + 1] && k > 0) k--;
                if (k > 0) {
                    for (r = m; v[k] > v[r]; r--);
                    swap(&v[k], &v[r]);
                    revert(k, m, (m - k) / 2);
                    arrNb++;
                    listArrang(m, v);
                }
            }
            while (k > 0);
        }
    }
}
```

Generating a power set

- We wish to generate all 2^n subsets of set $A=\{a_1,\ldots,a_n\}$ (power set)
  - All subsets of $A=\{a_1,\ldots,a_\mathrm{n}\}$ can be divided to two groups
    - Ones that contain a_n
    - Ones that does not contain a_n
  - Ones that does not contain a_n are all subsets of $\{a_1,\ldots,a_{n-1}\}$
    - When we have all subsets of $\{a_1,\ldots,a_{n-1}\}$ we can create all subsets of $\{a_1,\ldots,a_n\}$ by adding all elements with a_n inserted

Generating a power set

- Again, we try to generate all subsets without generating power sets of smaller sets
- We can associate 2^n subsets of n elements set $A=\{a_1,\ldots,a_n\}$ with 2^n bit strings $b_1\ldots b_n$
  - $b_i = 1$ if a_i is element of set
  - $b_i = 0$ if a_i is not element of set
- For 3 elements set $\{a_1,\ a_2,\ a_3\}$
  - 000 – the empty set
  - 111 – the set itself
  - 110 – subset $\{a_1,\ a_2\}$
- We can create bit strings by generating binary numbers from 0 to 2^n-1
- Example:

Bit strings	000	001	010	011	100	101	110	111
Subsets	$\varnothing$	a_3	a_2	a_2, a_3	a_1	a_1, a_3	a_1, a_2	a_1, a_2, a_3

Generating a power set. Non recursive

```
void allSubsetsNR(int n)
{
   int i, setNb, p[MAXN]; // p[i]=1 if element i is a member,
                           // i.e p is a characteristic vector
   setNb=1;
   for (i=1; i<=n; i++) p[i]=0; /*empty set */
   listSet(n, setNb, p);
   for (i=n; i>0; ) /* generate next subsets */
   {
      if (p[i] == 0)
      {
         p[i] = 1;
         setNb++;
         listSet(n, setNb, p);
         i = n;
      }
      else
      {
         p[i] = 0;
         i--;
      }
   }
}
```

Generating a power set. Recursive

```
void allSubsetsRec(int n, int i)
{
   int j;
   for (j=0; j <= 1; j++)
   {
      p[i] = j;
      if (i < n)
         allSubsetsRec(n, i+1);
      else
      {
         setNb++;
         listSet(n, setNb);
      }
   }
}
```

```
void listSet(int n, int
   setNb)
{
   int i;

  printf("\n%3d { ", setNb);
   for (i = 1; i <=n; i++)
      if (p[i] == 1 )
         printf(" %2d,", i);
   printf("\b  }");
   if ( setNb % 20 == 0 )
   getch();
}
```

- Note that array `p` and variable `setNb` must be global

Reading

- Deitel: chapter 11
- Prata: chapter 13
- King: chapter 22
- Supplemental:
  - R. Sedgewick: http://www.cs.princeton.edu/~rs/talks/perms.pdf

Summary

- File handling
  - High level I/O
    - fopen
    - fclose
    - fread
    - fwrite
    - fsetpos, fgetpos, ftell, fseek
- Applications
  - Combinatorial generation: generating subsets
    - Cross product (Cartesian product)
    - Combinations
    - Permutations
    - Arrangements
    - Power set