Additional Capabilities

Chapter 14

Conditional Operator
An unusual operator

- ?:  
  - Only tertiary operator (three operands)  
  - Behaves like an if-statement that produces a value  
  - Use if
    - it all fits on one line  
    - it should produce a value  
    - else use an if-statement  
  - (exp1) ? (exp2) : (exp3)
    - if exp1 is non-zero, the value of the conditional expression is exp2
    - if exp1 is zero, the value of the conditional expression is exp3
    - expressions are placed in parentheses if more than a constant or variable
  - max = (x > y) ? x : y;
  - count = (count == max) ? 0 : (count + 1);
  - index = (index+1 == size) ? 0 : ++index;

typedef Statements
Creating type aliases

- A typedef statement creates an alias or new name for an existing type (i.e., does not create a new type)
  - A “shorthand” notation
  - More mnemonic or self documenting
  - Improve portability by creating machine-dependent types (see <sys/types.h>)
- General format: typedef old_name new_name;
  - typedef unsigned long ulong;
  - typedef int boolean;
  - typedef unsigned int size_t;
  - ulong counter;
  - size_t length;

typedef and structs
A common short cut

typedef struct list            /* stuff in blue is the old type */
{  char* name;
   struct list* link;
 } LIST;

struct list head;             /* without typedef */
LIST head;                    /* with typedef */

Conditional Preprocessor Directives
Supporting multiple platforms

#ifdef __TURBOC    /* 2 under scores */
  # include <conio.h>   /* Borland */
#endif

#ifdef __TURBOC
  clrscr();            /* Borland */
#else
  if (_MSC_VER)
    system("cls");    /* Microsoft */
#else
  system("clear");   /* Unix */
#endif

Bit-Shifting Operations
Shifts all bits in the left operand

- Shifts the bits in the left operand by the amount specified by the right operand
  - Each operand must be an integer (char, short, int, or long)
  - << 11001100 << 2 ? 00110000
  - each bit shifted is equivalent to × 2
  - >> Exact behavior is implementation dependent
    - 11001100 >> 2 ? 11100001 signed and sign extend
    - 10011110 >> 2 ? 00100001 unsigned or w/o sign extend
    - 11001110 >> 2 ? 00110001 signed, unsigned, or w/o sign extend
    - Left hand operand is often unsigned to avoid the ambiguity
    - each bit shifted is equivalent to × 2
Bitwise Operators

Effects all bits in the integral operands

\[
\begin{align*}
\& \quad & (\text{bitwise and}) & 1100 \\
& & \& 1001 & 1000 \\
\mid & (\text{bitwise or}) & 1100 \\
& & \mid 1001 & 1101 \\
\text{often used with a mask} & & & \\
\end{align*}
\]

Often used with a mask

\[
\begin{align*}
\wedge \quad & (\text{exclusive or}) & 1100 \\
& & \wedge 1001 & 0101 \\
\sim & (\text{complement}) & 1100 \\
& & \sim 0011 & 0011 \\
\text{these operations are reversible} & & & \\
\end{align*}
\]

Bitwise AND

Masking out bits

- The bitwise AND operator is used to mask out or switch off bits

\[
\begin{align*}
\text{data} & = 10110110 \\
\text{mask} & = 00011111 \\
\text{result} & = 00001110 \\
\end{align*}
\]

Bitwise OR

Setting bits

- Bitwise OR is used set or switch on bits (similar to adding)

```c
#define EXECUTE 0x1
#define WRITE 0x2
#define READ 0x4

mode = READ | WRITE | EXECUTE;

modes = ((READ | WRITE) << 6) | (READ << 3) | READ;
```

Masks

```c
#define S_IFREG 0100000 /* regular file */
#define S_IFDIR 0040000 /* directory */
#define S_IFCHR 0020000 /* character device */
#define S_IFIFO 0010000 /* fifo */
#define S_ISUID 0004000 /* set UID bit */
#define S_ISGID 0002000 /* set GID bit */
#define S_ISVTX 0001000 /* sticky bit */
#define S_IRUSR 0400 /* owner has read permission */
#define S_IWUSR 0200 /* owner has write permission */
#define S_IXUSR 0100 /* owner has execute permission */
#define S_IRGRP 0040 /* group has read permission */
#define S_IWGRP 0020 /* group has write permission */
#define S_IXGRP 0010 /* group has execute permission */
#define S_IROTH 0004 /* others have read permission */
#define S_IWOTH 0002 /* others have write permission */
#define S_IXOTH 0001 /* others have execute permission */
```

struct VS union

Allocating and interpreting memory

```c
struct demo {
    char c;
    int i;
    double d;
    char* s;
};

union constant {
    char c;
    int i;
    double d;
    char* s;
};
```

stat Structure

One / file system entry

```c
struct stat {
    dev_t st_dev; /* device */
    ino_t st_ino; /* inode */
    mode_t st_mode; /* protection */
    nlink_t st_nlink; /* number of hard links */
    uid_t st_uid; /* user ID of owner */
    gid_t st_gid; /* group ID of owner */
    dev_t st_rdev; /* device type */
    off_t st_size; /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for filesystem */
    blkcnt_t st_blocks; /* number of blocks allocated */
    time_t st_atime; /* time of last access */
    time_t st_mtime; /* time of last modification */
    time_t st_ctime; /* time of last status change */
};
```
**st_mode**

unsigned short (16 bits)

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
                  |    |
                  SUID  SGID  sticky
                owner  group  others
```

- character device
- directory
- regular file

**Compressed Data Extraction**

Bit fields and unions

```c
struct modes {
    unsigned int others : 3;
    unsigned int group : 3;
    unsigned int user : 3;
    unsigned int type : 7;
};

union short_to_bits {
    mode_t statmode;
    struct modes conv;
};
```