

# Enumerations

# Review

- two dimensional array
  - `char *p[2] = {"MB", "Mercedes Bentz"};`

```
int main(int argc, char *argv[])
{
    int    i;

    printf("argc = %d\n", argc);
    for (i = 0; i < argc; ++i)
        printf("argv[%d] = %s\n", i, argv[i]);
    return 0;
}
```

# Review – function pointer

```
double sum_square(double f(double), int m, int n)
{
    int    k;
    double sum = 0.0;

    for (k = m; k <= n; ++k)
        sum += f(k) * f(k);
    return sum;
}
```

# Bit Operations

- most CPU processes data in word unit
  - 32 bits are processed in parallel
- if you need just one bit of information, the remaining space for 31bits is wasted

# C Bitwise Operators

<b>logical operators</b>	<b>(unary) bitwise complement</b>	<b>~</b>
	<b>bitwise and</b>	<b>&amp;</b>
	<b>bitwise exclusive or</b>	<b>^</b>
	<b>bitwise (inclusive) or</b>	<b> </b>
<b>shift operators</b>	<b>left shift</b>	<b>&lt;&lt;</b>
	<b>right shift</b>	<b>&gt;&gt;</b>

# Bitwise Complement

- binary representation of int a = 70707
  - 00000000 00000001 00010100 00110011
- $\sim a$ 
  - 11111111 11111110 11101011 11001100
  - decimal representation = -70708

# 2's Complement

Value of n	Binary representation	Bitwise complement	Two's complement representation of -n	Value of -n
7	00000000 00000111	11111111 11111000	11111111 11111001	-7
8	00000000 00001000	11111111 11110111	11111111 11111000	-8
9	00000000 00001001	11111111 11110110	11111111 11110111	-9
-7	11111111 11111001	00000000 00000110	00000000 00000111	7

# Bitwise Logical Operators

Declarations and initializations		
<code>int a = 33333, b = -77777;</code>		
Expression	Representation	Value
<code>a</code>	00000000 00000000 10000010 00110101	333333
<code>b</code>	11111111 11111110 11010000 00101111	-77777
<code>a &amp; b</code>	00000000 00000000 10000000 00100101	32805
<code>a ^ b</code>	11111111 11111110 01010010 00011010	-110054
<code>a   b</code>	11111111 11111110 11010010 00111111	-77249
<code>~(a   b)</code>	00000000 00000001 00101101 11000000	77248
<code>(~a &amp; ~b)</code>	00000000 00000001 00101101 11000000	77248



# Shift Operators

Declarations and initializations

```
char c = 'Z';
```

Expression	Representation	Action
<code>c</code>	00000000 00000000 00000000 01011010	unshifted
<code>c &lt;&lt; 1</code>	00000000 00000000 00000000 10110100	left-shifted 1
<code>c &lt;&lt; 4</code>	00000000 00000000 00000101 10100000	left-shifted 4
<code>c &lt;&lt; 31</code>	00000000 00000000 00000000 00000000	left-shifted 31

## Declarations and initializations

```
int      a = 1 << 31; /* shift 1 to the high bit */
unsigned b = 1 << 31;
```

Expression	Representation	Action
a	10000000 00000000 00000000 00000000	unshifted
a >> 3	11110000 00000000 00000000 00000000	right-shifted 3
b	10000000 00000000 00000000 00000000	unshifted
b >> 3	00010000 00000000 00000000 00000000	right-shifted 3

# Masks

```
void bit_print(int a)
{
    int i;
    int n = sizeof(int) * CHAR_BIT;      /* in limits.h */
    int mask = 1 << (n - 1);            /* mask = 100...0 */

    for (i = 1; i <= n; ++i) {
        putchar(((a & mask) == 0) ? '0' : '1');
        a <<= 1;
        if (i % CHAR_BIT == 0 && i < n)
            putchar(' ');
    }
}
```

# Packing

```
/* Pack 4 characters into an int. */
```

```
#include <limits.h>
```

```
int pack(char a, char b, char c, char d)
```

```
{
```

```
    int p = a;          /* p will be packed with a, b, c, d */
```

```
    p = (p << CHAR_BIT) | b;
```

```
    p = (p << CHAR_BIT) | c;
```

```
    p = (p << CHAR_BIT) | d;
```

```
    return p;
```

```
}
```

# Unpacking

```
/* Unpack a byte from an int. */

#include <limits.h>

char unpack(int p, int k)                /* k = 0, 1, 2, or 3 */
{
    int      n = k * CHAR_BIT;           /* n = 0, 8, 16, or 24 */
    unsigned mask = 255;                 /* low-order byte */

    mask <<= n;
    return ((p & mask) >> n);
}
```

```
k = 1, n = 8
p           = 01100001 01100010 01100011 01100100
mask        = 00000000 00000000 11111111 00000000
p & mask    = 00000000 00000000 01100011 00000000
return      00000000 00000000 00000000 01100011
```

# Enumeration Types

```
enum day {sun, mon., tue, wed, thu, fri, sat};
```

```
enum day d1, d2, holidays;
```

- enumerators are of int type
- the first one is zero, ..

```
enum suit {club = 3, diamonds, hearts = 0; spades} a, b;
```

```

enum day find_next_day(enum day d)
{
    enum day next_day;
    switch (d) {
    case su:
        next_day = mo;
        break;
    case mo:
        next_day = tu;
        break;
    ...
    case sa:
        next_day = su;
        break;
    }
    return next_day;
}

```

Enumerators are constants of type `int`. Thus, they can be used in case labels in a switch statement.

*Example of using enumerators with switch*

```
enum day {su, mo, tu, we, th, fr, sa};
```

# Enumeration Types

- type casting applies

```
enum day {su, mo, tu, we, th, fr, sa};
```

```
enum day find_next_day(enum day d)
{
    assert((int) d >= 0 && (int) d < 7)
    return ((enum day)((int) d + 1) % 7);
}
```



# Preprocessor

- works before compilation

`#include`

- includes “file name” or
- search <file name> in other directories

- macros

```
#define SEC_PER_DAY (60 * 60 * 24)
```

```
#define SQ(x) ((x) * (x))
```

```
#define max(x, y) (((x) > (y)) ? (x) : (y))
```

# Conditional Compilation

```
#if win
    #define WIN32
    #include <windows.h>
    #include <winsock.h>
#else
    #define closesocket close
    #include <sys/types.h>
    #include <sys/socket.h>
    #include <netinet/in.h>
    #include <arpa/inet.h>
    #include <netdb.h>
#endif

#include <stdio.h>
#include <string.h>
.....
```

set the value of win using  
#define

# Other directives

- defined operator

```
# if (defined(HP) || defined(SUN)) && !defined(SUN3)
```

```
.....
```

```
#endif
```

- debugging code

```
# if DEBUG
```

```
.....
```

```
#endif
```

```
#if HEAP_SIZE < 24000
```

```
    # error "not enough space for malloc"
```

```
#endif
```

# Stringization

```
#define message(a, b) printf (#a "and " #b);
```

message(Hail, SKKU) is replaced by  
printf("Hail" "and" "SKKU")

```
#define X(i)  x ## i
```

X(2) is replaced by X2