Review Topics

- Binary Representation:
  - Binary numbers, signed int, floating point
  - ASCII
  - Bitwise operations
- C operators, structures
- Pointers, * and &, arrays, struct, typedef
- Dynamic memory allocation, linked lists
- Functions, Stack-frames, recursion
- I/O: stdin/stdout, formatting, file
Converting Decimal to Binary (2’s C)

First Method: Division

1. Find magnitude of decimal number. (Always positive.)
2. Divide by two – remainder is least significant bit.
3. Keep dividing by two until answer is zero, writing remainders from right to left.
4. Append a zero as the MS bit; if original number was negative, take two’s complement.

<table>
<thead>
<tr>
<th>Division</th>
<th>Remainder</th>
<th>Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>104/2</td>
<td>52</td>
<td>0</td>
</tr>
<tr>
<td>52/2</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>26/2</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>13/2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6/2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3/2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1/2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ X = 104_{\text{ten}} \]

\[ X = 01101000_{\text{two}} \]
Converting Decimal to Binary (2’s C)

Second Method: Subtract Powers of Two

1. Find magnitude of decimal number.
2. Subtract largest power of two less than or equal to number.
3. Put a one in the corresponding bit position.
4. Keep subtracting until result is zero.
5. Append a zero as MS bit; if original was negative, take two’s complement.

\[ \begin{array}{c|c|c}
 n & 2^n \\
 0 & 1 \\
 1 & 2 \\
 2 & 4 \\
 3 & 8 \\
 4 & 16 \\
 5 & 32 \\
 6 & 64 \\
 7 & 128 \\
 8 & 256 \\
 9 & 512 \\
 10 & 1024 \\
\end{array} \]

\[ X = 104_{\text{ten}} \]

\[ 104 - 64 = 40 \text{ \ bit 6} \]
\[ 40 - 32 = 8 \text{ \ bit 5} \]
\[ 8 - 8 = 0 \text{ \ bit 3} \]

\[ X = 01101000_{\text{two}} \]
Number Representation
Binary to Hexadecimal Conversion

• Method: Group binary digits, convert to hex digits using table.
• Question: What is binary 011001111000100111111011011010 in hexadecimal?

0110 0111 1000 1001 1111 1110 1101 1010
6 7 8 9 F E D A

• Answer: 0x6789FEDA
Number Representation

Two’s Complement

- Invert all the bits, and add 1.
- Question: What is the value -8 in (16-bit) 2’s complement form?

\[ 8 = 0x0008 = 0000 \ 0000 \ 0000 \ 1000 \]

Invert bits \[ 1111 \ 1111 \ 1111 \ 0111 \]

Add one \[ + \ 0000 \ 0000 \ 0000 \ 0001 \]

Answer \[ = 1111111111111100 \] (binary)

Answer \[ = 0xFFF8 \]

Answer: \[ 0xFFF8 = -8 \] decimal
Two’s Complement

Two’s complement representation developed to make circuits easy for arithmetic.

- for each positive number \(X\), assign value to its negative \((-X)\), such that \(X + (-X) = 0\) with “normal” addition, ignoring carry out

\[
\begin{align*}
00101 & \quad (5) & \quad 01001 & \quad (9) \\
+ 11011 & \quad (-5) & + & \quad \underline{_____} & \quad (-9) \\
00000 & \quad (0) & \quad 00000 & \quad (0)
\end{align*}
\]
Number Representation

**Binary to Floating Point Conversion**

- Single-precision IEEE floating point number:
  \[
  \begin{array}{c c c}
  1 & 01111110 & 10000000000000000000000 \\
  \end{array}
  \]

- Or 0xBF400000
- Sign is 1 – number is negative.
- Exponent field is 01111110 = 126 – 127 = -1 (decimal).
- Fraction is \(1.100000000000\ldots\) = 1.5 (decimal).
- Value = \(-1.5 \times 2^{(126-127)} = -1.5 \times 2^{-1} = -0.75\)
Floating Point to Binary Conversion

- Value = 4.25
  - Number is positive – sign is 0
  - Fraction is 100.01 (binary), normalize to 1.0001 * 2^2
  - Exponent is 2 + 127 = 129 (decimal) = 10000001

- Single-precision IEEE floating point number:
  
  \[
  \begin{array}{cccccccccccccccccccc}
  \text{sign} & \text{exponent} & \text{fraction} \\
  0 & 10000001 & 00010000000000000000000000000000 \\
  \end{array}
  \]

  - or 0x40880000

MR1-9
Operators

bitwise operators:
\[
\begin{align*}
\text{int } i &= 0x11223344; \\
\text{int } j &= 0xFFFF0000; \\
\text{printf(}"0x%x\n", \sim i); \quad 0xEEDDCCBB \\
\text{printf(}"0x%x\n", i \& j); \quad 0x11220000 \\
\text{printf(}"0x%x\n", i | j); \quad 0xFFFF3344
\end{align*}
\]

logical operators: (in C 0 is false, everything else is true!)
\[
\begin{align*}
\text{int } i &= 0x11223344; \\
\text{int } j &= 0x00000000; \\
\text{printf(}"0x%x\n", \sim i); \quad 0x00000000 \\
\text{printf(}"0x%x\n", i \&\& j); \quad 0x00000000 \\
\text{printf(}"0x%x\n", i || j); \quad 0x00000001
\end{align*}
\]

arithmetic operators:
\[
\begin{align*}
\text{int } i &= 10; \\
\text{int } j &= 2; \\
\text{printf(}"d\n", i + j); \quad 12 \\
\text{printf(}"d\n", i - j); \quad 8 \\
\text{printf(}"d\n", i \ast j); \quad 20 \\
\text{printf(}"d\n", i \big/ j); \quad 5
\end{align*}
\]
C Programming

Control Structures

C conditional and iterative statements

- if statement
  
  ```
  if (value == 0x12345678)
      printf("value matches 0x12345678\n");
  ```

- for loop
  
  ```
  for (int i = 0; i < 8; ++i)
      printf("i = %d\n", i);
  ```

- while loop
  
  ```
  int j = 6;
  while (j--)
      printf("j = %d\n", j);
  ```
Functions in C

double ValueInDollars(double amount, double rate);

main()
{
    ... 
    dollars = ValueInDollars(francs, DOLLARS_PER_FRANC);
    printf("%f francs equals %f dollars.\n", francs, dollars);
    ...
}

double ValueInDollars(double amount, double rate)
{
    return amount * rate;
}
Pointers: * and & operators

```c
int i;
int *ptr;
i = 4;
ptr = &i;
*ptr = *ptr + 1;
```

store the value 4 into the memory location associated with i
store the address of i into the memory location associated with ptr
read the contents of memory at the address stored in ptr
store the result into memory at the address stored in ptr
## Relationship between Arrays and Pointers

An array name is essentially a pointer to the first element in the array:

```c
char word[10];
char *cptr;
cptr = word;  /* points to word[0] */
```

**Difference:**
Can change the contents of `cptr`, as in:

```c
cptr = cptr + 1;
```

(The identifier "word" is not a variable.)

**Correspondence:**

<table>
<thead>
<tr>
<th>cptr</th>
<th>word</th>
<th>&amp;word[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(cptr + n)</td>
<td>word + n</td>
<td>&amp;word[n]</td>
</tr>
<tr>
<td>*cptr</td>
<td>*word</td>
<td>word[0]</td>
</tr>
<tr>
<td>*(cptr + n)</td>
<td>*(word + n)</td>
<td>word[n]</td>
</tr>
</tbody>
</table>
C pointers and arrays

```c
void foo(int *pointer)
{
    *(pointer+0) = pointer[2] = 0x1234;
    *(pointer+1) = pointer[3] = 0x5678;
}

int main(int argc, char *argv[])
{
    int array[] = {0, 1, 2, 3};
    foo(array);
    for (int i = 0; i <= 3; ++i)
        printf("array[%d] = %x\n", i, array[i]);
}
```
C strings

```c
char *string = "hello";
char *carray = { 'h', 'e', 'l', 'l', 'o' };
char label[20];
strcpy(label, "hello");
strcat(label, " world");
printf("%s\n", label); hello world
printf("%d\n", strlen(label)); 11
printf("%d\n", sizeof(label)); 20
```
C data structures

// Structure definition
struct sCoordinate
{
    float X;
    float y;
    float z;
};

typedef struct {
    ... 
} Coordinate;
C data structures

// Structure allocation
struct sCoordinate coordinates[10]; // no typedef
typedef Coordinate coordinates[10]; // typedef
Coordinate *coordinates = malloc(sizeof(Coordinate)*10);

// Structure access
coordinates[5].X = 1.0f;
pCoordinate->X = 1.0f;
Dynamic memory Allocation

planes = (Flight*) malloc(n* sizeof(Flight));

.. 
newNode->next = nextNode;

   means (*newNode).next = nextNode

.. 
free(newNode);
Scanning the linked List of structs

Car *ScanList(Car *head, int searchID)
{
    Car *previous, *current;
    previous = head;
    current = head->next;
    /* Traverse until ID >= searchID */
    while ((current!=NULL) && (current->vehicleID < searchID)) {
        previous = current;
        current = current->next;
    }
    return previous;
}
int a = 100;        int b = 65;        char c = 'z';
char banner[10] = "Hola!";    double pi = 3.14159;

printf("The variable 'a' decimal: %d\n", a);
printf("The variable 'a' hex: %x\n", a);
printf("The variable 'a' binary: %b\n", a);
printf("'a' plus 'b' as character: %c\n", a+b);
printf("A char %c.\t A string %s\n A float %f\n", c, banner, pi);

char name[100];    int bMonth, bDay, bYear;
double gpa;

scanf("%s %d/%d/%d %lf", 
       name, &bMonth, &bDay, &bYear, &gpa);
The type of a stream is a "file pointer", declared as:

```
FILE *infile;
```

The `fopen` function associates a physical file with a stream.

```
FILE *fopen(char* name, char* mode);
```

Once a file is opened, it can be read or written using `fscanf()` and `fprintf()`, respectively.

```
fprintf(outfile, "The answer is %d\n", x);

fscanf(infile, "%s %d/%d/%d %lf",
       name, &bMonth, &bDay, &bYear, &gpa);
```
```c
int RS (int n) {
    if (n == 1)
        return 1;
    else
        return n + RS (n - 1);
}

int main () {
    res = RS (4);
    return value = 10
}
```

**Recursion**

```
res = RS (4);
```

```
return 4 + RS (3);
```

```
return 3 + RS (2);
```

```
return 2 + RS (1);
```

```
return 1;
```

RS (4)

RS (3)

RS (2)

RS (1)