C Language II

CMSC 313
Sections 01, 02

C Input/Output

stdin, stdout, stderr

C opens three input/output devices automatically:

stdin
The "standard input" device, usually your keyboard
stdout
The "standard output" device, usually your monitor
stderr
The "standard error" device, usually your monitor

Some C library I/O functions automatically use these devices

Formatted Console Output

- `printf( )` outputs formatted text to `stdout`

```
printf( format, arg1, arg2, … );
```

- Example:

```
int n = 3 ;
printf ("Value = %d\n", n) ;
```

- `format` is a string containing
  - conversion specifications
  - literals to be printed

Adapted from Richard Chang, CMSC 313 Spring 2013

printf( ) Conversions

Conversions specifications begin with % and end with a conversion character.

Between the % and the conversion character MAY be, in order A minus sign specifying left-justification

The minimum field width

A period separating the field width and precision The precision that specifies

Maximum characters for a string

Number of digits after the decimal for a floating point Minimum number of digits for an integer

An h for "short" or an l (letter ell) for long

man printf for more documentation.

Adapted from Richard Chang, CMSC 313 Spring 2013

Common printf( ) Conversions

Spec	Conversion performed
`%d`	print integer as a decimal number (base 10)
`%u`	print integer as unsigned number
`%s`	print string
`%f`	print double as a floating point number
`%x`	print integer in hexadecimal (base 16)
`%c`	print integer as ASCII character
`%p`	print pointer in hexadecimal (implementation dependent)

Adapted from Richard Chang, CMSC 313 Spring 2013

printf( ) Examples

```
int anInt = 5678;
double aDouble = 4.123;
#define NAME "Bob"

/* what is the output from each printf( ) */
printf("%d is a large number\n", anInt);
printf("%8d is a large number\n", anInt);
printf("%-8d is a large number\n", anInt);
printf("%10.2f is a double\n", aDouble);
printf("The sum of %d and %8.4f is %12.2f\n",
    anInt, aDouble, anInt + aDouble);
printf ("Hello %s\n", NAME);
```


Formatted Output Example

Use field widths to align output in columns

```
int i;
for (i = 1 ; i < 5; i++)
  printf("%2d %10.6f %20.15f\n", i, sqrt(i), sqrt(i));
```

```
12 1234567890 12345678901234567890
 1   1.000000    1.000000000000000
 2   1.414214    1.414213562373095
 3   1.732051    1.732050807568877
 4   2.000000    2.000000000000000
```


Keyboard Input

- `scanf` reads user input from `stdin`.
- Syntax for `scanf( )` is similar to `printf( ) scanf( format, arg1, arg2, ... )`
- The format string similar structure to `printf( )`.
- The arguments must be *addresses* of the variables.

scanf( ) Format String

- The `scanf( )` format string usually contains conversion specifications that tell `scanf( )` how to interpret the next "input field". An input field is a string of non-whitespace characters.
- The format string usually contains:
  - Blanks or tabs which are ignored
  - Ordinary characters which are expected to match the next (non- whitespace) character input by the user
  - Conversion specifications usually consisting
    - % character indicating the beginning of the conversion
    - An optional h, l (ell) or L
    - A conversion character which indicates how the input field is to be interpreted.

Adapted from Richard Chang, CMSC 313 Spring 2013

Common scanf( ) Conversions

Spec	Conversion performed
`%d`	a decimal (integer) number
`%u`	An unsigned decimal (integer) number
`%x`	a hexadecimal number
`%f`	a floating point number with optional sign, decimal point, and exponent
`%s`	a string delimited by white space, NOT an entire line
`%c`	a single character (possibly a whitespace char)

Adapted from Richard Chang, CMSC 313 Spring 2013

scanf( ) Examples

```
int age;
double gpa;
char initial;

printf("input your middle initial: ");
scanf("%c", &initial);     // Note &
printf("Input your age: ");
scanf("%d", &age);
printf("input your gpa: ");
scanf("%lf", &gpa);
```

Adapted from Richard Chang, CMSC 313 Spring 2013

Unix I/O redirection

- Redirect input (read from `infile` instead of keyboard):
  `a.out < infile`
- Redirect output (write to `outfile` instead of screen):
  `a.out > outfile`
- Redirect both:
  `a.out < infile > outfile`
- Redirect `stdout` and `stderr` to `outfile`
  `a.out >& outfile`
- Redirect `stdout` to `outfile` and `stderr` to `errfile`
  `(a.out > outfile) >& errfile`

Adapted from Richard Chang, CMSC 313 Spring 2013

Text File I/O

- Use `fprintf()` and `fscanf()` functions instead of `printf()` and `scanf()`.
- Must open file before reading/writing: `fopen()`
- Must close file after all done: `fclose()`
- Use file handle to specify file.
- File handle returned by `fopen()`:

```
FILE *myFile ;
myFile = fopen ("bob.txt", "r") ;
if (myFile == NULL) {
    /* handle the error */
}
```

Adapted from Richard Chang, CMSC 313 Spring 2013

fopen()

`fopen( )` requires two parameters

1. The name of the text file to be opened
2. The text file open "mode"
   - `"r"` open the file for reading only
   - `"w"` create the file for writing; delete existing file
   - `"a"` append; open or create the file for writing at the end
   - `"r+"` open the file for reading and writing
   - `"w+"` create the file for reading & writing; deletes existing file
   - `"a+"` open or create the file for reading or writing at the end

Adapted from Richard Chang, CMSC 313 Spring 2013

fscanf.c

```
#include <stdio.h>
#include <stdlib.h>  /* for "exit" */
int main ( )
{
  double x ;
  FILE *ifp ;

  /* try to open the file for reading, check if successful */
  /* if it wasn't opened exit gracefully */
  ifp = fopen("test_data.dat", "r") ;
  if (ifp == NULL) {
    printf ("Error opening test_data.dat\n");
    exit (-1);
  }
  fscanf(ifp, "%lf", &x) ;     /* read one double from the file */
  fclose(ifp);                 /* close the file when finished  */

  /* check to see what you read */
  printf("x = %.2f\n", x) ;
  return 0;
}
```

Adapted from Richard Chang, CMSC 313 Spring 2013

Detecting end-of-file with fscanf

- When reading an unknown number of data elements from a file using **`fscanf( )`**, we need a way to determine when the file has no more data to read, i.e, we have reached the "end of file".
- Fortunately, the return value from **`fscanf( )`** holds the key. **`fscanf( )`** returns an integer which is the number of data elements read from the file. If end-of-file is detected the integer return value is the special value **`EOF`**

Adapted from Richard Chang, CMSC 313 Spring 2013

EOF Example Code

```
/* code snippet that reads an undetermined number of
   integer student ages from a file and prints them out
   as an example of detecting EOF (scanfEOF.c)
*/
FILE *inFile;
int age;

inFile = fopen( "myfile", "r" );
if (inFile == NULL) {
    printf ("Error opening myFile\n");
    exit (-1);
   }

while ( fscanf(inFile, "%d", &age ) != EOF )
  printf( "%d\n", age );

fclose( inFile );
```

Adapted from Richard Chang, CMSC 313 Spring 2013

fprintf.c

```
/* fprintf.c */
#include <stdio.h>
#include <stdlib.h>  /* exit */
int main ( )
{
  double pi = 3.14159 ;
  FILE *ofp ;

  /* try to open the file for writing, check if successful */
  /* if it wasn't exit gracefully */
  ofp = fopen("test.out", "w") ;
  if (ofp == NULL) {
    printf ("Error opening test.out\n");
    exit (-1);
  }

  /* write to the file using printf formats */
  fprintf(ofp, "Hello World\n");
  fprintf(ofp, "PI is defined as %6.5lf\n", pi);

  fclose(ofp);            /* close the file when finished reading */

  return 0;
}
```


Characters & Strings

char Type

- C supports the `char` data type for storing a single character.
- `char` uses one byte of memory
- `char` constants are enclosed in single quotes

```
char myGrade = 'A';
char yourGrade = '?';
```

Special Characters

Use \ for escape sequences.

For example

- \n is the newline character
- \t is the tab character
- \" is the double quote (necessary since double quotes are used to enclose strings
- \' is the single quote (necessary since single quotes are used to enclose chars
- \\ is the backslash (necessary since \ now has special meaning
- \a is beep which is unprintable

Special Char Example Code

- What is the output from these statements?

```
printf("\t\tMove over\n\nWorld, here I come\n");
```

```
        Move over

World, here I come
```

```
printf("I\'ve written \"Hello World\"\n\t many times\n\a");
```

```
I've written "Hello World"
        many times
<BEEP>
```


Character Library Functions

int isdigit (int c);
Determine if c is a decimal digit ('0' - '9')
int isxdigit(int c);
Determines if c is a hexadecimal digit ('0' - '9', 'a' - f', or 'A' - 'F')
int isalpha (int c);
Determines if c is an alphabetic character ('a' - 'z' or 'A- 'Z')
int isspace (int c);
Determines if c is a whitespace character (space, tab, etc)
int isprint (int c);
Determines if c is a printable character
int tolower (int c);
int toupper (int c);
Returns c changed to lower- or upper-case respectively, if possible

Character Library Functions

Include header file use character library functions:

```
#include <ctype.h>
```

Technically functions take an `int` parameter, not `char`.

Return type is also `int` . 0 = False, not 0 = True.

`man ctype.h` for more functions and complete documentation.

Character Input/Output

- Use `%c` in `printf( )` and `fprintf( )` to output a single character.

```
char yourGrade = 'A';
printf( "Your grade is %c\n", yourGrade);
```

- Input char(s) using `%c` with `scanf( )` or `fscanf( )`

```
char grade, scores[3];
```

  - %c inputs the next character, which may be whitespace
    scanf("%c", &grade);
  - %nc inputs the next n characters, which may include whitespace
    scanf("%3c", scores); // note – no & needed

Strings in C

- String = null terminated array of char.
- null = `'\0'`
- String constants in double quotes are null terminated.
- Strings do not "know" their own length.
- Initialization:

```
char name4[20] = {'B', 'o', 'b', 'b', 'y', '\0'};
char name5[6] = "Bobby";  // this is NOT assignment
char name6[ ] = "Bobby";
```

String Output

```
char name[ ] = “Bobby Smith”;
printf( “My name is %s\n”, name);

char book1[ ] = “Flatland”;
char book2[ ] = “Brave New World”;

// Minimum field width, and right- and left-justify
printf (“My favorite books are %12s and %12s\n”, book1, book2);
printf (“My favorite books are %-12s and %-12s\n”, book1, book2);
```


Dangerous String Input

- Why is the following dangerous?

```
char name[22];
printf(“ Enter your name: “);
scanf( “%s”, name);
```

- Long names will overwrite memory.

Safer String Input

```
char name[22];
printf(“ Enter your name: “);
scanf( “%21s”, name);  // Will stop after 21 (note: not 22) chars
                       // (need last byte for ‘\0’
```

C String Library

- C provides a library of string functions.
- To use the string functions, include `<string.h>`.
- Some of the more common functions are listed here on the next slides.
- To see all the string functions, type `man string.h` at the unix prompt.

C String Library (2)

- Must `#include <string.h>`
- These functions look for the '\0' character to determine the end and size of the string
  - strlen(const char string[ ])
    - Returns length of string, not counting '\0'
  - strcpy(char s1[ ], const char s2[ ])
    - Copies s2 on top of s1. Must have enough space in s1!!!
    - The order of the parameters mimics the assignment operator
  - strcmp (const char s1[ ] , const char s2[ ])
    - Returns < 0, 0, > 0 if s1 < s2, s1 == s2 or s1 > s2 lexigraphically
  - strcat(char s1[ ] , const char s2[ ])
    - Appends (concatenates) s2 to s1. Must have enough space in s1!!!

C String Library (3)

- Some safer versions from the C string library have an additional size parameter.
  - strncpy(char s1[ ], const char s2[ ], int n)
    - Copies at most n characters of s2 on top of s1.
    - Does not null terminate s1 if length of s2 >= n !!!
    - The order of the parameters mimics the assignment operator
  - strncmp (const char s1[ ] , const char s2[ ], int n)
    - Compares up to n characters of s1 with s2
    - Returns < 0, 0, > 0 if s1 < s2, s1 == s2 or s1 > s2 lexigraphically
  - strncat(char s1[ ], const char s2[ ] , int n)
    - Appends at most n characters of s2 to s1

String Code

```
char first[10] = "bobby";
char last[15] = "smith";
char name[30];
char you[ ] = "bobo";

strcpy( name, first );
strcat( name, last );
printf( "%d, %s\n", strlen(name), name );

strncpy( name, last, 2 );
printf( "%d, %s\n", strlen(name), name );

int result = strcmp( you, first );
result = strncmp( you, first, 3 );
strcat( first, last );
```

Adapted from Richard Chang, CMSC 313 Spring 2013

Simple Encryption

```
char c, msg[] = "this is a secret message";
int i = 0;
char code[26] =     /* Initialize our encryption code */
 {'t','f','h','x','q','j','e','m','u','p','i','d','c',
'k','v','b','a','o','l','r','z','w','g','n','s','y'} ;

printf ("Original phrase: %s\n", msg);

/* Encrypt */
while( msg[i] != '\0' ){
   if( isalpha( msg[ i ] ) ) {
      c = tolower( msg[ i ] ) ;
      msg[ i ] = code[ c - 'a' ] ;
   }
   ++i;
}
printf("Encrypted: %s\n", msg ) ;
```

Adapted from Richard Chang, CMSC 313 Spring 2013

Arrays of Strings

- **An initialized array of string constants**

```
char months[ ][4] = {
    "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
};
int m;
for ( m = 0; m < 12; m++ )
   printf( "%s\n", months[ m ] );
```

- **Alternative: use typedef**

```
typedef char Acronym[4];
Acronym months[ ] = {
    "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
};
```

Adapted from Richard Chang, CMSC 313 Spring 2013

sprintf()

- sprintf( ) works just like printf( ) or fprintf( ), but puts its "output" into the specified character array.
- The character array must be big enough.

```
char message[ 100 ];
int myAge = 4;
sprintf( message, "I am %d years old\n", age);
printf( "%s\n", message);
```


Structs

C++ vs C

- Suppose you were assigned a write an application about points and straight lines in a coordinate plane.
- In C++, you'd correctly design a Point class and a Line class using composition.
- What about in C?

No Classes in C

- **Because C is not an OOP language, there is no way to combine data and code into a single entity.**
- **C does allow us to combine related data into a structure using the keyword `struct`.**
- **All data in a `struct` variable can be accessed by any code.**
- **Think of a `struct` as an OOP class in which all data members are public, and which has no methods.**

Adapted from Richard Chang, CMSC 313 Spring 2013

No Classes in C

- Because C is not an OOP language, ~~there is no way to combine data and code into a single entity~~.
- Related data and functions form an "Abstract Data Type." Accessibility is enforced by a programmer's good judgment and not by the compiler.
- C does allow us to combine related data into a structure using the keyword `struct`.
- All data in a `struct` variable can be accessed by any code.
- Think of a `struct` as an OOP class in which all data members are public, and which has no methods.

Adapted from Richard Chang, CMSC 313 Spring 2013

`struct` Definition

- **The general form of a structure definition is**

```
struct tag
{
    member1_declaration;
    member2_declaration;
    member3_declaration;
     . . .
    memberN_declaration;
};
```

Note the semi-colon

where `struct` is the keyword, `tag` names this kind of `struct`, and `member_declarations` are variable declarations which define the members.

Adapted from Richard Chang, CMSC 313 Spring 2013

C `struct` Example

- Defining a `struct` to represent a point in a coordinate plane

```
struct point ←———— point is the struct tag
{
   int x;    /* x-coordinate */
   int y;    /* y-coordinate */
};
```

- Given the declarations

```
struct point p1;
struct point p2;
```

we can access the members of these `struct` variables:

* the x-coordinate of p1 is
* the y-coordinate of p1 is
* the x-coordinate of p2 is
* the y-coordinate of p2 is

Using `struct` Members

```
int main ( )
{
   struct point leftEendPt, rightEndPt, newEndPt;

   printf("Left end point cooridinates ");
   scanf( "%d %d", &leftEendPt.x, &leftEndPt.y);

   printf("Right end point's x-coordinate: ");
   scanf( "%d %d", &rightEendPt.x, &rightEndPt.y);

   // add the endpoints
   newEndPt.x = leftEndPt.x + rightEndPt.x;
   newEndPt.y = leftEndPt.y + rightEndPt.y;

   // print new end point
   printf("New endpoint (%2d, %2d)\n", newEndPt.x,
   newEndPt.y);

   return 0;
}
```


Initializing a `struct`

```
struct point middle = { 6, -3 };
```

is equivalent to:

```
struct point middle;
middle.x = 6;
middle.y = -3;
```

`struct` Variants

```
struct point {
  int x, y;
} endpoint, upperLeft;
```

defines the structure named `point`

AND

declares the variables `endpoint` and `upperLeft` to be of this type.

`struct` + typedef

```
typedef struct point {
    int x, y;
} POINT;
```

defines the structure named `point` and defines `POINT` as a `typedef` (type alias) for the `struct`.

```
    POINT upperRight;
```

is now equivalent to:

```
    struct point endpoint;
```


`struct` Assignment

```
struct point p1;
struct point p2;

p1.x = 42;
p1.y = 59;

p2 = p1;  /* structure assignment copies members */
```

The values of `p2`'s members are the same as `p1`'s members.
E.g. `p1.x = p2.x = 42` and `p1.y = p2.y = 59`

`struct` within a `struct`

```
typedef struct line
{
  POINT leftEndPoint;
  POINT rightEndPoint;
} LINE;

LINE line1, line2;
line1.leftEndPoint.x = 3;
line1.leftEndPoint.y = 4;
```


Arrays of `struct`

```
LINE lines[5];  // or struct line lines[5]

printf("%d\n", lines[2].leftEndPoint.x);
```


Arrays within a `struct`

- **Structs may contain arrays as well as primitive types**

```
struct month
{
  int nrDays;
  char name[ 3 + 1 ];
};

struct month january = { 31, "JAN"};
```

A Bit More Complex

```
struct month allMonths[ 12 ] = {
  {31, "JAN"}, {28, "FEB"}, {31, "MAR"},
  {30, "APR"}, {31, "MAY"}, {30, "JUN"},
  {31, "JUL"}, {31, "AUG"}, {30, "SEP"},
  {31, "OCT"}, {30, "NOV"}, {31, "DEC"}
};
// write the code to print the data for September
printf( "%s has %d days\n",
  allMonths[8].name, allMonths[8].nrDays);

// what is the value of allMonths[3].name[1]?
```


Size of a `struct`

- As with primitive types, we can use sizeof( ) to determine the number of bytes in a struct

```
int pointSize = sizeof( POINT );
int lineSize = sizeof (struct line);
```

As we'll see later, the answers may surprise you!