

## today

- homework #1 — due on monday sep 23, 6am
- some miscellaneous topics:
  - logical operators
  - random numbers
  - character handling functions
  - FILE I/O
- strings
- arrays
- pointers

## logical operators (1)

- in C are the same as in Java

meaning	C operator
AND	&&
OR	
NOT	!

- since there are no *boolean* types in C, these are mainly used to connect clauses in *if* and *while* statements
- remember that
  - non-zero  $\Rightarrow$  *true*
  - zero  $\Rightarrow$  *false*

## logical operators (2)

- exercise: what is the output of the following code fragment?

```
int n = 12345, m = 0;
printf( "n and n=[%d]\n", n && n );
printf( "n or n=[%d]\n", n || n );
printf( "not n=[%d]\n", !n );
printf( "n and m=[%d]\n", n && m );
printf( "n or m=[%d]\n", n || m );
printf( "not m=[%d]\n", !m );
```

## logical operators (3)

- there are also *bitwise* operators in C, in which each bit is an operand:

meaning	C operator
bitwise AND	&
bitwise OR	

- example:

```
int a = 8; /* this is 1000 in base 2 */
int b = 15; /* this is 1111 in base 2 */
```

	1000 (=8)		1000 (=8)
a & b $\Rightarrow$	$\&$ 1111 (=15)	a   b $\Rightarrow$	$ $ 1111 (=15)
	1000 (=8)		1111 (=15)

## logical operators (4)

- exercise: what is the output of the following code fragment?

```
int a = 12, b = 7;
printf( "a && b = %d\n", a && b );
printf( "a || b = %d\n", a || b );
printf( "a & b = %d\n", a & b );
printf( "a | b = %d\n", a | b );
```

## random numbers (1)

- with computers, nothing is random (even though it may seem so at times...)
- there are two steps to using random numbers in C:
  1. seeding the random number generator
  2. generating random number(s)
- standard library function:

```
#include <stdlib.h>
```
- seed function:

```
srand( time ( NULL ) );
```
- random number function returns a number between 0 and RAND\_MAX (which is  $2^{32}$ )

```
int i = rand();
```

## random numbers (2)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main( void ) {
    int r;
    srand( time ( NULL ) );
    r = rand() % 100;
    printf( "pick a number between 0 and 100...\n" );
    printf( "was %d your number?", r );
}
```

## character handling functions (1).

- character handling library

```
#include <ctype.h>
```
- digit recognition functions (bases 10 and 16)
- alphanumeric character recognition
- case recognition/conversion
- character type recognition
- these are all of the form:

```
int isdigit( int c );
```

where the argument *c* is declared as an *int*, but it is interpreted as a *char* so if *c* = '0' (i.e., the ASCII value '0', index=48), then the function returns *true* (non-zero int) but if *c* = 0 (i.e., the ASCII value NULL, index=0), then the function returns *false* (0)

## character handling functions (2).

### digit recognition functions (bases 10 and 16)

- `int isdigit( int c );`  
returns *true* (i.e., non-zero int) if `c` is a decimal digit (i.e., in the range `'0' . . . '9'`);  
returns 0 otherwise
- `int isxdigit( int c );`  
returns *true* (i.e., non-zero int) if `c` is a hexadecimal digit (i.e., in the range `'0' . . . '9', 'A' . . . 'F'`); returns 0 otherwise

## character handling functions (3).

### alphanumeric character recognition

- `int isalpha( int c );`  
returns *true* (i.e., non-zero int) if `c` is a letter (i.e., in the range `'A' . . . 'Z', 'a' . . . 'z'`); returns 0 otherwise
- `int isalnum( int c );`  
returns *true* (i.e., non-zero int) if `c` is an alphanumeric character (i.e., in the range `'A' . . . 'Z', 'a' . . . 'z', '0' . . . '9'`); returns 0 otherwise

## character handling functions (4).

### case recognition

- `int islower( int c );`  
returns *true* (i.e., non-zero int) if `c` is a lowercase letter (i.e., in the range `'a' . . . 'z'`);  
returns 0 otherwise
- `int isupper( int c );`  
returns *true* (i.e., non-zero int) if `c` is an uppercase letter (i.e., in the range `'A' . . . 'Z'`);  
returns 0 otherwise

### case conversion

- `int tolower( int c );`  
returns the value of `c` converted to a lowercase letter (does nothing if `c` is not a letter or if `c` is already lowercase)
- `int toupper( int c );`  
returns the value of `c` converted to an uppercase letter (does nothing if `c` is not a letter or if `c` is already uppercase)

## character handling functions (5).

### character type recognition

- `int isspace( int c );`  
returns *true* (i.e., non-zero int) if `c` is a space; returns 0 otherwise
- `int iscntrl( int c );`  
returns *true* (i.e., non-zero int) if `c` is a control character; returns 0 otherwise
- `int ispunct( int c );`  
returns *true* (i.e., non-zero int) if `c` is a punctuation mark; returns 0 otherwise
- `int isprint( int c );`  
returns *true* (i.e., non-zero int) if `c` is a printable character; returns 0 otherwise
- `int isgraph( int c );`  
returns *true* (i.e., non-zero int) if `c` is a graphics character; returns 0 otherwise

## character handling functions (6).

- exercise:  
start with the following code fragment that loops through the extended ASCII character set (0..255) and prints out each index and each ASCII value:

```
int i;
for ( i=0; i<256; i++ ) {
    printf( "%d %c\n", i, i );
}
```

make this into a program

call each ctype function shown on the previous slides that start with is and print out the return values — then you can see which characters are printable, graphics, etc.

## file I/O (1).

- file handling involves three steps:
  1. opening the file
  2. reading from and/or writing to the file
  3. closing the file
- files in C are *sequential access*
- think of it as a cursor that sits at a position in the file
- with each read and write operation, you move that cursor's position in the file
- the last position in the file is called the "end-of-file" and is typically written as: <EOF>
- all the functions described on the next few slides are defined in the <stdio.h> header file

## file I/O (2).

### opening files

- `FILE *fopen( const char *filename, const char *mode );`
- `filename` is a string containing the name of the file you want to open; this file is in the current working directory or else you have to include a full path specification
- `mode` is one of the following:

mode	meaning	cursor position	create file?
r	read only	beginning of file	no
r+	read/write	beginning of file	no
w	write only	beginning of file	yes
w+	read/write	beginning of file	yes
a	write only	end of file	no
a+	read/write	end of file	no

the last column indicates whether the file is created if it does not exist — this is only done with the w modes

- the function returns a value of type `FILE *`, which is a *file pointer* (we'll talk about pointers later today), or `NULL` if there is an error

## file I/O (3).

### reading from and writing to files

- these functions are just like `printf` and `scanf`, except that instead of writing to the screen and reading from the keyboard, they write to and read from a file
- for writing to a file:

```
int fprintf( FILE *fp, const char *format /*, args...*/ );
```

this function returns the number of bytes written  
`fp` is the file pointer of the file you are writing to

- for reading from a file:

```
int fscanf( FILE *fp, const char *format /*, args...*/ );
```

this function returns the number of bytes read  
`fp` is the file pointer of the file you are reading from

## file I/O (4).

### closing files

- `int close( FILE *fp );`  
`fp` is the pointer to the file you want to close (the value returned from a previous call to `fopen`)

## strings (1).

- storing multiple characters in a single variable
- data type is still `char`
- BUT it has a *length*
- last character the is *terminator*: `'\0'`, aka `NULL`
- string constants are surrounded by *double* quotes: `"`
- example:

```
char s[6] = "ABCDE";
```

## strings (2).

- example:  
`char s[6] = "ABCDE";`
- storage looks like this: 

A	B	C	D	E	\0
---	---	---	---	---	----
- so with strings, you really only access the values stored at indices 0 through *length* - 2, since the value stored at *length* - 1 is always `\0`

## strings (3).

- printing strings
- format sequence: `%s`
- example:

```
#include <stdio.h>
int main( void ) {
    char str[6] = "ABCDE";
    printf( "str = %s\n", str );
} /* end of main() */
```

- output:  
ABCDE

## strings (4).

- string handling library

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

- functions include:

```
int strlen( char *s );
```

this function returns the number of characters in `s`; note that this is NOT the same thing as the number of characters allocated for the string array

- `int strcmp( const char *s1, const char *s2 );`

“This function returns an integer greater than, equal to, or less than 0, if the string pointed to by `s1` is greater than, equal to, or less than the string pointed to by `s2` respectively. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of bytes that differ in the strings being compared.”

- for more information and more string functions, do:

```
unix$ man strcmp
```

## arrays (1).

- a string is an *array* of characters
- an array is a “regular grouping or ordering”
- a data structure consisting of related elements of the same data type
- in C, an array has a length associated with it
- arrays need:
  - data type
  - name
  - length
- length can be determined:
  - *statically* — at compile time  
e.g., `char str1[10];`
  - *dynamically* — at run time  
e.g., `char *str2;`

## arrays (2).

- defining a variable is called “allocating memory” to store that variable
- defining an array means allocating memory for a group of bytes, i.e., assigning a label to the first byte in the group
- individual array elements are *indexed*
  - starting with 0
  - ending with *length* – 1
- indices follow array name, enclosed in square brackets (`[ ]`)  
e.g., `arr[25]`

## array (3).

character array example

```
#include <stdio.h>
#define MAX 6
int main( void ) {
    char str[MAX] = "ABCDE";
    int i;
    for ( i=0; i<MAX-1; i++ ) {
        printf( "%c", str[i] );
    }
    printf( "\n" );
} /* end of main() */
```

## arrays (4).

integer array example

```
#include <stdio.h>
#define MAX 6
int main( void ) {
    int arr[MAX] = { -45, 6, 0, 72, 1543, 62 };
    int i;
    for ( i=0; i<MAX; i++ ) {
        printf( "%d", arr[i] );
    }
    printf( "\n" );
} /* end of main() */
```

## pointers (1).

- variables that contain memory addresses as their values
- other data types we've learned about in C use *direct* addressing
- pointers facilitate *indirect* addressing
- declaring pointers:
  - pointers indirectly address memory where data of the types we've already discussed is stored (e.g., int, char, float, etc.)
  - declaration uses asterisks (\*) to indicate a pointer to a memory location storing a particular data type
- example:

```
int *count;
float *avg;
```

## pointers (2).

- ampersand & is used to *dereference* a pointer
- it says: return the address of the variable argument
- example:

```
int count = 12;
int *countPtr = &count;
```
- &count returns the *address* of count and stores it in the pointer variable countPtr
- a picture:

```
countPtr   count
  ●         →  12
```

## pointers (3).

here's another example:

```
int i = 3, j = -99;
int count = 12;
int *countPtr = &count;
```

and here's what the memory looks like:

variable name	memory location	value
count	0xbffff4f0	12
i	0xbffff4f4	3
j	0xbffff4f8	-99
...		
countPtr	0xbffff600	0xbffff4f0
...		

#### pointers (4).

- an array is some number of contiguous memory locations
- an array definition is really a pointer to the starting memory location of the array
- and pointers are really integers
- so you can perform integer arithmetic on them
- e.g., +1 increments a pointer, -1 decrements
- you can use this to move from one array element to another

#### pointers (5).

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    int i, *j, arr[5];
    srand( time ( NULL ));
    for ( i=0; i<5; i++ )
        arr[i] = rand() % 100;
    printf( "arr=%p\n",arr );
    for ( i=0; i<5; i++ ) {
        printf( "i=%d arr[i]=%d &arr[i]=%p\n",i,arr[i],&arr[i] );
    }
    j = &arr[0];
    printf( "\nj=%p *j=%d\n",j,*j );
    j++;
    printf( "after adding 1 to j:\n j=%p *j=%d\n",j,*j );
}
```

#### pointers (6).

and the output is...

```
arr=0xbffff4f0
i=0 arr[i]=29 &arr[i]=0xbffff4f0
i=1 arr[i]=8 &arr[i]=0xbffff4f4
i=2 arr[i]=18 &arr[i]=0xbffff4f8
i=3 arr[i]=95 &arr[i]=0xbffff4fc
i=4 arr[i]=48 &arr[i]=0xbffff500
```

```
j=0xbffff4f0 *j=29
after adding 1 to j:
j=0xbffff4f4 *j=8
```

#### today's example.

*The tendency of people to focus on the meaning of sentences influences their ability to notice some of the obvious features.*

exercise:

- write a program to count the number of "f"s in the above.
- write a program to count the total number of characters
- write a program to count the total number of words
- write a program to write this to a file
- write a program to read this from a file