Week 8 Lecture 3

Finishing up C

Conditional Operator

Conditional Operator

- Syntax
  - <condition> ? <expression1> : <expression2>
- Value
  - If condition is true, expression1, otherwise expresion2
- Example
  - 0 > x ? x : -x
    - Value: absolute value of x

Comma Operator

Comma operator

- Syntax
  - $<expr_1>$, $<expr_2>$, …, $<expr_n>$
- Value
  - $Expr_n$
  - Precedence
    - Lowest: $<expr_i>$ is evaluated before going to next expr.
- Example
  - for(i = 1, j = 2; i < 10; i++, j+=2)

Unions

Unions

- Unions save memory space by different types in a single variable
- Allows you to define a single super type for a set of sub types.
- The variable can only store on type at a time.

Unions

- Syntax:
  - union <tag> {dec1, …, decn} <vars>
- Syntax
  - Each var may have the type of the declaration

Union Example

```
typedef struct queen
{
  char display;
} queen;

typedef struct king
{
  char display;
} king;

typedef union piece
{
  queen q;
  king k;
} piece;
```

```
main()
{
  piece p;
  king k;
  queen q;

  k.display = 'k';
  q.display = 'q';

  p.k = k;
  printf("Piece 1: %c\n", p.k.display);
  p.q = q;
  printf("Piece 2: %c\n", p.q.display);
}
```

```
student> ./union
Piece 1: k
Piece 2: q
```

Storage Classes

Storage Class

- In addition to scope, lifetime and value, variables may have a storage class
  - Auto: on stack
  - Register: special memory location
  - Static: on heap rather than on stack
  - Extern: defined in separate compilation module

Storage Class

- In addition to scope, lifetime and value, variables may have a storage class
  - Auto: on stack
  - Register: special memory location
  - Static: on heap rather than on stack
  - Extern: defined in separate compilation module

Stack vs Heap

- Functions get new variables each time called
  - The new variables are on the stack
  - A stack is last in, first out
  - The default storage class for local variable is auto
- Global variable have only on instance
  - The variable is on the heap
  - The heap is shared by the entire program
  - The default storage class for globals is static

Automatic Variables

- Designated by auto keyword, though rarely used.
- Default class for variables (including parameters) in functions, including main().
- Default value for variables local to blocks (e.g. for (int i = 0; ...))
- Not initialize. Contain garbage values.

Static Variables

- Designated by static keyword
- Default class for global variables
- Only one version of the variable
  - Auto variables are create new for each new function
- Variables in functions may be declared static
  - This is the only common use of the keyword

External variables

- Designated by extern keyword
- External variables are defined in other object files (i.e., .o files)
- During linking the external variable are associated with the memory location defined in the other .o file.
- Used to expose variables in external modules.

Register Variables

- Designated by register keyword
- A register is a memory location on the Arithmetic-Logic Unit that does the calculations
  - Registers need to be loaded before arithmetic can be done.
  - Declaring a variable register removes that step.
- Don't user register variables
  - The compiler is better at deciding where variables should go than you do.

Bit operators

Bit Operators

- C lets you operate on the bit representation of an int or a char.
  - To have reasonable results, the type much be interpretable as a string of bits
- A char is 1 byte or 4 bits
  - -128 .. 127
  - 0 .. 256
- An int is 4 bytes or 32 bits
  - -2,147,483,648 to 2,147,483,647
  - 0 to 4,294,967,295

Bitwise Operator

Bitwise Operators

operator	coding	notes
left shift	<<	shifts LHS left by no. of bits in RHS
right shift	>>	shifts LHS right by no. of bits in RHS
bitwise AND	&	returns parrallel and of LHS and RHS input bits
bitwise OR	|	returns parrallel or of LHS and RHS input bits
exclusive OR	^	**Returns 1 if LHS and RHS bits are different, 0 if same**
twos complement	~	Unary NOT operator. Returns 0 for RHS 1 and 1 for 0.

Operating on Bits

Bitwise operators

- The shift operator:
  - x << n
    - Shifts the bits in x n positions to the left, shifting in zeros on the right.
    - If x = 1111 1111 1111 0000_2

    x << 1 equals 1111 1111 1110 0000_2
  - x >> n
    - Shifts the bits in x n positions right.
      - shifts in the sign if it is a signed integer (arithmetic shift)
      - shifts in 0 if it is an unsigned integer
    - x >> 1 is 0111 1111 1111 1000_2

Operating on Bits

- Bitwise logical operations
  - Work on all integer types
    - & Bitwise AND

      x&y
    - | Bitwise Inclusive OR

      x|y
    - ^ Bitwise Exclusive OR

      x^y output 1 only if input bits are different
    - ~ The complement operator

      ~ y Complements all of the bits of X

Bitwise Operator Example

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

int main(int argc, char *argv[])
{
  char x = 15;

  printf("A char is %d bits\n", CHAR_BIT);
  printf("An int is %ld bits\n\n",
         sizeof(int) * CHAR_BIT);

  printf("x: %d, x << 1: %d\n", x, x << 1);
  printf("x: %d, x >> 1: %d\n", x, x >> 1);
  printf("x: %d, x & 1: %d\n", x, x & 1);
  printf("x: %d, x | 1: %d\n", x, x | 1);
  printf("x: %d, x ^ 1: %d\n", x, x ^ 1);
  printf("x: %d, ~x: %d\n", x, ~x);
}
```

```
student> ./bits
A char is 8 bits
An int is 32 bits

x: 15, x << 1: 30
x: 15, x >> 1: 7
x: 15, x & 1: 1
x: 15, x | 1: 15
x: 15, x ^ 1: 14
x: 15, ~x: -16
```

Left shift, right shift

- We define char x = 15
  - 15 is 0000 1111 in binary
- Left shift moves the bits to the left, filling with 0
  - e.g. x << 1 -> 0001 1110 or 16 + 8 + 4 + 2 = 30
- Right shift moves the bits to the left
  - e.g. x >> 1 → 0000 0111 or 4 + 2 + 1 = 7

And, Or, Exclusive or, Not

- &&, || and ! operate on true value
- &, |, ^, and ~ work on bits

&&	true	false
true	true	false
false	false	false

||	true	false
true	true	true
false	true	false

!	
true	false
false	true

&	1	0
1	1	0
0	0	0

|	1	0
1	1	1
0	1	0

^	1	0
1	0	1
0	1	0

~	
1	0
0	1

&, |, ^, ~

- Remember x is 15_{10}, which is $0000\ 1111_2$
  - x & 1 -> 0000 0001 or 1
  - x | 1 → 0000 1111 or 8 + 4 + 2 + 1 = 15
  - x ^ 1 → 0000 1110 or 8 + 4 + 2 + 1 = 14
  - ~x → 1111 0000 or -16

1111 0000 is -16?

- Computers use *twos compliment* arithmetic
- Twos complement inverts the digits then adds 1
  - e.g. 15 is 0000 1111; -15 is 1111 0001
- Advantages
  - Can add positive and negative numbers
  - Only one 0

Twos Compliment

- Computed by ~x + 1
- Examples:

7 + -7 = 0

```
 0111
+1001
-----
 0000
```

5 + -4 = 0

```
 0101
+1100
-----
 0001
```

5 + -7 = -2

```
 0101
+1001
-----
 1110
```

Bits	Value
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

C String Library

The C String Library

- String functions are provided in an ANSI standard string library.
  - Access this through the include file:

    #include <string.h>
  - Includes functions such as:
    - Computing length of string
    - Copying strings
    - Concatenating strings
  - This library is guaranteed to be there in any ANSI standard implementation of C.

Strlen()

- This function counts and returns the number of characters in a string. The length does not include a null character.

Syntax n=strlen(string);

- Where n is integer variable. Which receives the value of length of the string.

Example

- length=strlen(“Hollywood”);
- The function will assign number of characters 9 in the string to a integer variable length.

/* Use of the function strlen( ) */

```
main( )
{
char arr[ ] = "Bamboozled" ;
int len1, len2 ;
len1 = strlen ( arr ) ;
len2 = strlen ( "Humpty Dumpty" ) ;
printf ( "\nstring = %s length = %d", arr, len1 ) ;
printf ( "\nstring = %s length = %d", "Humpty Dumpty", len2 ) ;
}
```

Comparing Strings

- C strings can be compared for equality or inequality
- If they are equal - they are ASCII identical
- If they are unequal the comparison function will return an int that is interpreted as:

  < 0 : str1 is less than str2

  0 : str1 is equal to str2

  > 0 : str1 is greater than str2

Strcmp()

int strcmp (char *str1, char *str2) ;

- Does an ASCII comparison one char at a time until a difference is found between two chars
  - Return value is as stated before
- If both strings reach a '\0' at the same time, they are considered equal.

Strcmp()

- Strcmp(string1,string2)
- string1 & string2 may be string variables or string constants.
- Some computers return a negative if the string1 is alphabetically less than the second and a positive number if the string is greater than the second.

Example:

- strcmp(“Newyork”,”Newyork”) will return zero because 2 strings are equal.

- strcmp(“their”,”there”) will return a -9 which is the numeric difference between ASCII ‘i’ and ASCII ’r’.

- strcmp(“The”, “the”) will return -32 which is the numeric difference between ASCII “T” & ASCII “t”.

Strcat()

- Included in string.h and comes in the form:

  strcat (str1, str2);

  - Appends a copy of str2 to the end of str1
  - A pointer equal to str1 is returned

  – Ensure that str1 has sufficient space for the concatenated string!

    - Array index out of range will be the most popular bug in your C programming career.

Example

```
#include <string.h>
main() {
    char str1[27] = "abc";
    char str2[100];
    printf("%d\n",strlen(str1));
    strcpy(str2,str1);
    puts(str2);
    puts("\n");
    strcat(str2,str1);
    puts(str2);
}
```

Strcpy()

- Strcpy () copies a string including the null character terminator from source string to destination .
- strcpy(destination string2, source string1);

```
#include <string.h>
 char string1[] = "Hello, world!";
 char string2[20];
 strcpy(string2, string1);
```

Usage of strcpy()

```
#include <string.h>
 main()
{
    /* Example 1 */
    char string1[ ] = "A string declared as an array.\n";
    /* Example 2 */
    char *string2 = "A string declared as a pointer.\n";
    /* Example 3 */
    char string3[30];
     strcpy(string3, "A string constant copied in.\n");
    printf (string1);
    printf (string2);
    printf (string3);
}
```

Example- To check whether a char is vowel or not

```
main()
{
char a[10];int i;
printf("enter string");
gets(a);
for(i=0;a[i]!='\0';i++)
{
        if(a[i]=='a'||a[i]=='e'||a[i]=='i'||a[i]=='o'||a[i]==a[i]=='u'||a[i]=='A'||a[i]=='E'||a[i]==
        'I'||a[i]=='O'||a[i]=='U')
        {
                printf("%c",a[i]);
        }
}
}
```

Example -To check whether string is palindrome or not

```
main()
{
char a[10];
int l,i,j;
clrscr();
printf("enter string");
gets(a);
l=strlen(a);
for(i=0,j=l-1;i<l/2;i++,j--)
{
        if(a[i]!=a[j])
        {
         printf("not palindrome");
         break;
        }
}
if(i==l/2)
{
printf("palindrome");
}

}
```