CMSC 11
Functions

Introduction

- Some programs may grow very large having thousand or even million lines of code (LOC)
- Some problems are too big to be solved in "one shot", it is better to break the big problem into smaller subproblems and provide a solution to the smaller subproblems
- Functions provide a mechanism to implement the solution to smaller subproblems
- Consider a program that will simulate a bank. This program should allow the user to create an account, withdraw, deposit, check balance, etc..

Functions in Math

- Definition of functions in math (from Wikipedia)
  A function associates a single output to each input element drawn from a fixed set, such as the real numbers
  A function f from a set X to a set Y associates to each element x in X an element $y=f(x)$ in Y.
  Example: Let $y = f(x) = x^2$
  - x is the independent variable
  - y is the dependent variable
  - the domain of f(x) is the set of possible values of x
  - the range of f(x) is the set of possible values of y
- How do we implement in C functions in math?

Function Definition

- "Let $y = f(x) = x^2$" is a function definition
- the *name* of the function is f
- the *parameter* of the function is x
- the *body* of the function is x^2
- The domain and range are the set of integers
- The function definition in C is given below

```
int f(int x)
{
     int retval;
     retval = x * x;
     return (retval);
}
```

In general the syntax for function definition is:

```
<return type> <function name>(<formal parameter list>)
{
     <local variable definitions>
     <statements>
     <return statement>;
}
```

- <return type> – int
- <function name> – f
- <formal parameter list> – <int x>
- <local variable definitions> – int retval;
- <statements> – retval = x * x;
- <return statement> = return (retval);

- When a function will return nothing use `void` for the <return type>
- In C, we always define a function called `main()`. This function is the entry point of all C programs and execution of statements starts at this function.
- In essence, programming in C involves defining functions!

Function Call

- After a function has been defined, we can call/invoke the function
- Function call example

```
int main(){
     f(5);
}
```

In general the syntax for function call is:

```
<function name>(<actual parameter list>);
```

- <function name> – f
- <actual parameter list> – <5>

- In the above example, we did not store the result of the function call! We should assign the result to a variable using an assignment statement.

```
int main(){
     int y = f(5);
}
```

- The value of `y` after the function call will be `25`

- During a function call, control is transferred to the *called function (f)*. That is, the *statements specified in the called function definition are executed*. Then control is transferred back to the *calling function(main).* Recall that the assignment statement has a left hand side(lhs) and a right hand side(rhs). The rhs is evaluated first and the result of the evaluation is stored on the lhs.
- In essence, *a function call is an expression* because it evaluates to a value, the *return value.*
- Functions you define are called *user-defined* functions. C has a collection of functions called the *C Standard Library*
- You can call user-defined functions inside other user-defined functions.

More Examples

```
int sumOfSquares(int x, int y){
     int retval;
     retval = f(x) + f(y);
     return retval;
}

int factorial(int n){
     int retval=1;
     int i;
     for (i=1; i <= n; i++){
          retval = retval * i;
     }
     return retval;
}

void greet(){
     printf(“Hello!\n”);
}
```

Parameters

- *Formal parameters* are *specified* during function definition
  Each formal parameter has the syntax <data type> <name>
  Formal parameters are treated like local variables defined inside the function
- *Actual parameters* are *passed* during function call
  Each actual parameter is an expression (literal, variable, function call) that evaluates to a value of the type specified in the formal parameter in the function definition
  When control is transferred to the called function, a *local copy* (inside the called function) of the actual parameters is made. The local copy is what is used during computations inside the called function. This is called *pass by value*
  The following code shows that the value of a remains unchanged even

after the call to `addOne()`

```
void addOne(int x){
     x = x + 1;
}
int main(){
     int a=5;
     printf(“a: %d\n”,a);
     addOne(a);
     printf(“a: %d\n”,a);
}
```

In order to achieve the desired effect, we can use *pass by reference*

```
void addOne(int *x){
     *x = *x + 1;
}
int main(){
     int a=5;
     printf(“a: %d\n”,a);
     addOne(&a);
     printf(“a: %d\n”,a);
}
```