Lecture 14: Functions (Cont.)
Function Definition!

return-type function-name ( parameters )
{
    declarations
    statements
}

• Develop a function that adds two integer values and return the result value.

```c
int add(int a, int b)    //define
{
    int c;
    c = a + b;
    return c;
}

main()
{
    int x = 3;
    int y = 5;
    int z;
    z = add(x, y);          //call
    printf("%d",z);
}
Develop a function that prints welcome message N times, where N is an input from the user.

```c
void display_welcome(int n)
{
    int i=0;
    for(i=0;i<n;i++){
        printf("welcome !! \n");
    }
    printf("done !! \n");
}

main()
{
    display_welcome(4);
}
```
• Develop a program that tests whether the integer user input number is prime or not.

• Your program uses a function named is_prime that returns true if the input is prime and false otherwise.

• is_prime divides its input n by each of the numbers between 2 and the n−1; if the remainder is ever 0, n isn’t prime.
#include <stdio.h>

bool is_prime(int n) {
    if (n <= 1) { //test the special cases
        return false;
    }
    int divisor;
    for (divisor = 2; divisor < n; divisor++) {
        if (n % divisor == 0)
            return false;
    }
    return true;
}

main(void) {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    if (is_prime(n))
        printf("Prime
");
    else
        printf("Not prime
");
}
Function Declarations

- C doesn’t require that the definition of a function *precede its calls*.
- However, calling a function before defining it, the compiler starts *making assumptions* (that may be wrong) and the program *won’t work as expected*.

```c
main(void) {
    double x=8, y=10, z;
    z = average(x, y);
}

int average(int a, int b) {
    return (a + b) / 2;
}
```
• One way to avoid the problem of *call-before-definition* is to arrange the program so that the definition of each function *precedes all its calls*, which *doesn’t always exist* and makes the program hard to understand (unnatural order).

• A better solution, is to *declare the function* before calling it.

• Such declaration informs the compiler about the *signature of the function*, whose *full definition* will appear *later*!
General form of a function declaration:

```
return-type function-name ( parameters ) ;
```
#include <stdio.h>

double average(double a, double b);

main(void)
{
    double x=8, y=10, z;

    z = average(x, y);
}

double average(double a, double b)
{
    return (a + b) / 2;
}
Pass array to function

```c
int sum_array(int a[], int n);

main(void) {
    int b[100], total;
    ... // fill the array with user-inputs

    total = sum_array(b, 100);
}

int sum_array(int a[], int n) {
    int i, sum = 0;
    for(i=0 ; i<n ; i++)
        sum +=a[i];

    return sum;
}
```
Function Arguments

• In C, arguments (parameters of the function) are passed by value.

• When a function is called, each argument is evaluated and its value is assigned to the corresponding parameter.

• Changes made to the parameters during the execution of the function don’t affect the value of the argument.
#include <stdio.h>

void update_value(int n){
    n = n * 2;
}

main(void)
{
    int X = 100;
    update_value(X);
}
Recursive Functions

• A function is recursive if it calls itself, example to compute $n! = n \times (n-1) \times (n-2)\times \ldots \times 1$

```c
int factorial(int n){
    if(n <= 1)
        return 1;
    else
        return ( n * factorial (n-1));
}
```
• Develop a function that *swaps* two integer values.

• Develop a function that returns the *average* of the minimum and maximum for the input integer array.