CSE 220 - C Programming
Fall 2015
Functions
Functions
Functions are modules of code that accomplish a specific task.
Functions are modules of code that accomplish a specific task.

(a positive number)

\[ \text{sqrt}(n) \]
Functions are modules of code that accomplish a specific task.

\[ \text{pow}(a, b) \]

\[ a^b \]
Categories of Functions in C

Basically there are **two categories** of functions:

**Predefined functions**: (Standard Library Functions) available in C standard library such as `stdio.h`, `math.h`, `string.h` etc.

**User-defined functions**: functions that programmers themselves implement for a specific task (which probably have not been defined in the standard libraries)
Predefined functions

**stdio.h**
- `printf()`
- `scanf()`

**math.h**
- `double pow(double a, double b)`
- `double sqrt(double a)`
- `double ceil(double a)`
- `double floor(double a)`
- `double log(double a)`
- `double log10(double a)`
- `double exp(double a)`
- `double cos(double a)`
- `double sin(double a)`
- `double tan(double a)`


\[
\begin{align*}
\text{pow}(a, b) & \quad \text{Exponentiation} \\
\sqrt{a} & \quad \text{Square root} \\
\lceil a \rceil & \quad \text{Ceiling} \\
\lfloor a \rfloor & \quad \text{Floor} \\
\log(a) & \quad \text{Logarithm} \\
\log_{10}(a) & \quad \text{Logarithm base 10} \\
\exp(a) & \quad \text{Exponential} \\
\cos(a) & \quad \text{Cosine} \\
\sin(a) & \quad \text{Sine} \\
\tan(a) & \quad \text{Tangent}
\end{align*}
\]
User-defined Functions in C
Finding Maximum Number in Two Arrays

Write a C program that finds the maximum number in two arrays

Array 1

4  6  7  8  9  -1

Array 2

2  3  12 -4  0  6  7  13  2  -3

Output: 13
Finding a Maximum Number in Two Arrays

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

int main()
{
    int arr1[6] = {4, 6, 7, 8, 9, -1};
    int arr2[10] = {2, 3, 12, -4, 0, 6, 7, 13, 2, -3};
    int max1, max2;

    /* find the maximum in the first array */
    max1 = arr1[0];
    for(int i = 1; i < 6; i++)
        if(arr1[i] > max1) 
            max1 = arr1[i];

    /* find the maximum in the second array */

    return 0;
}
```
Finding a Maximum Number in Two Arrays

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

int main()
{
    int arr1[6] = {4, 6, 7, 8, 9, -1};
    int arr2[10] = {2, 3, 12, -4, 0, 6, 7, 13, 2, -3};

    int max1, max2;

    /* find the maximum in the first array */
    max1 = arr1[0];
    for(int i = 1; i < 6; i++)
        if(arr1[i] > max1)
            max1 = arr1[i];

    /* find the maximum in the second array */
    max2 = arr2[1];
    for(int j = 1; j < 10; j++)
        if(arr2[j] > max2)
            max2 = arr2[j];

    return 0;
}
```

Maximum in the second array
Finding a Maximum Number in Two Arrays

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

int main()
{
    int arr1[6] = {4, 6, 7, 8, 9, -1};
    int arr2[10] = {2, 3, 12, -4, 0, 6, 7, 13, 2, -3};

    int max1, max2;

    /* find the maximum in the first array */
    max1 = arr1[0];
    for(int i = 1; i < 6; i++)
        if(arr1[i] > max1)
            max1 = arr1[i];

    /* find the maximum in the second array */
    max2 = arr2[1];
    for(int j = 1; j < 10; j++)
        if(arr2[j] > max2)
            max2 = arr2[j];

    if (max1 > max2)
        printf("The maximum is: %d\n", max1);
    else
        printf("The maximum is: %d\n", max2);

    return 0;
}
```
But ...

– We have *re-used the same piece of code* for determining a maximum number in the second array

• Copy & Paste?

We need an *abstraction* to resolve these issues: 
the concept of functions
Function

input array

size of array

max of an array

maximum

How to use?
Use `findMaxArray()` Function

```c
int main()
{
    int arr1[6] = {4, 6, 7, 8, 9, -1};
    int arr2[10] = {2, 3, 12, -4, 0, 6, 7, 13, 2, -3};

    int max1 = findMaxArray(arr1, 6);
    int max2 = findMaxArray(arr2, 10);

    if (max1 > max2)
        printf("The maximum is: %d\n", max1);
    else
        printf("The maximum is: %d\n", max2);

    return 0;
}
```
Function

If no-one gives us this function, how can we implement this function ourself?

findMaxArray(,)
Defining a Function: Header
To define a function we need to decide about 3 parts first:

1. Output type
2. Name
3. Input(s) type

```c
return_type function_name (type_1 parameter_1, type_2 parameter_2, ...)
{
    Body of function;
}
```
Defining a Function

The first line of every function definition is called **function header or signature**. It has 3 components as shown below:

```
return_type function_name (type_1 parameter_1, type_2 parameter_2, ...)
{
    Body of function;
}
```
**return_type**: Specifies the data type that the function should return to the caller program. Can be any of C data types: `char`, `float`, `int`, `long`, `double`, etc.
The first line of every function definition is called **function header or signature**. It has 3 components as shown below:

```plaintext
return_type function_name (type_1 parameter_1, type_2 parameter_2, ...)
{
    Body of function;
}
```
Defining a Function

**return_type:** Specifies the data type that the function should return to the caller program. Can be any of C data types: char, float, int, long, double, pointers etc.

**function_name:** Can have any name as long as the rules for C variable names are followed and must be unique.
Defining a Function

The first line of every function definition is called **function header or signature**. It has 3 components as shown below:

- The type of result the function is supposed to produce if any
- The unique name of the function
- The list of parameters with their types if any

```
return_type function_name (type_1 parameter_1, type_2 parameter_2, ...)
{
    Body of function;
}
```

**Example:**

```
float percentage (int a, int b)
```
Defining a Function

**return_type:** Specifies the data type that the function should return to the caller program. Can be any of C data types: char, float, int, long, double, pointers etc.

**function_name:** Can have any name as long as the rules for C variable names are followed and must be unique.

**parameter_list:** Many functions use arguments, the value passed to the function when it is called. A function needs to know the data type of each argument.
How to define a Function

```
return_type function_name (type_1 parameter_1, type_2 parameter_2, ...
{
    Body of function;
}
```
Defining a Function: Body
Defining a Function: Body

\[
\text{return}_\text{type } \text{function}_\text{name} \ (\text{type}_1 \ \text{parameter}_1, \ \text{type}_2 \ \text{parameter}_2, \ \ldots)
\]

\{

\text{Body of function}

\text{parameter}_1 \ \text{parameter}_2 \ \ldots

\text{a data with return}_\text{type}

\}

- **Function body**, containing the statements, which the function will perform, should begin with an opening brace and end with a closing brace.

- Enclosed in **curly braces** \{\}, immediately follows the function header.

- Real work in the program is done here.
Example

float `percentage` (int `a`, int `b`)
{
    float `p` = a/b*100;
    return `p`;
}

• Must specify:
  – Return type: `float`
  – function name: `percentage`
  – function parameters and type: `int a, int b`
  – function body
Keep in Mind ...

- The type must be listed for every parameter

```c
void add(int x, y)
{
    printf("%d + %d = %d\n", x, y, x+y);
}
```

```c
void add(int x, int y)
{
    printf("%d + %d = %d\n", x, y, x+y);
}
```
Complete the Finding Maximum Number by defining a Function

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

int findMaxArray(int arr[], int size)
{
    /* Add your code here */
}

int main()
{
    /* Add your code here */
    return 0;
}
```
Complete the Finding Maximum Number by defining a Function

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

int findMaxArray(int arr[], int size)
{
    // Your code here
}

int main()
{
    int arr1[6] = {4, 6, 7, 8, 9, -1};
    int arr2[10] = {2, 3, 12, -4, 0, 6, 7, 13, 2, -3};
    int max1 = findMaxArray(arr1, 6);
    int max2 = findMaxArray(arr2, 10);

    if (max1 > max2)
        printf("The maximum is: %d\n", max1);
    else
        printf("The maximum is: %d\n", max2);
    return 0;
}
```
Complete the Finding Maximum Number by defining a Function

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

int findMaxArray(int arr[], int size)
{
    int max = arr[0];
    for(int i = 1; i < size; i++)
        if(arr[i] > max)  
            max = arr[i];
    return max;
}

int main()
{
    int arr1[6] = {4, 6, 7, 8, 9, -1};
    int arr2[10] = {2, 3, 12, -4, 0, 6, 7, 13, 2, -3};

    int max1 = findMaxArray(arr1, 6);
    int max2 = findMaxArray(arr2, 10);

    if(max1 > max2)
        printf("The maximum is: %d\n", max1);
    else
        printf("The maximum is: %d\n", max2);
    return 0;
}
```
By using functions...

```c
#include "...."
...
... functionA();
... functionB();
... functionC();
...

int main()
{

}
```

Split your code into multiple independent tasks (functions) instead of a big chunk of a code.

Use the main as the driver of your whole program!
Why Functions?

• Building blocks of C programs
• Divide program into smaller pieces
• Easier to understand
• Easier to maintain
• Reuse code and avoid repetition
The Main Characteristics of a Function

- Named with a **unique** name
- Performs a **specific** task
- **Independent**
- May receive values from the calling program (caller)
- May return a value to the calling program
Defining a Function

No output parameter?

Functions don’t have to return anything: Return type is void

```c
void function_name (type_1 parameter_1, type_2 parameter_2, ...) {
    Body of function;
}
```
Defining a Function

No **output** parameter?

Functions don’t have to return anything: Return type is **void**

```c
void function_name (type_1 parameter_1, type_2 parameter_2, ...) {
    Body of function;
}
```

No **input** argument?

Functions don’t have to take input parameters: use **void** in place of parameters

```c
return_type function_name (void) {
    Body of function;
}
```
void sayHello(void)
{
    printf(“Hello everyone\n”);
}

Defining a Function
void print_happy_birthday( int age )
// This is the return type (void means no value is computed and returned by the function!)
{
    printf("Congratulations on your %d th Birthday\n", age);
    return;
    // you can "terminate" a void function by using return.
    // HERE it is REDUNDANT because the function is over anyway.
}
Defining a Function

How about **multiple outputs**?
Be patient, later in the course

How about **arrays**?
Functions can not return arrays.
Be patient, later in the course we will learn a simple trick to do this :-)
Calling a Function
Calling a Function

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

int findMaxArray(int arr[], int size)
{
    int max = arr[0];
    for(int i = 1; i < size; i++)
        if(arr[i] > max)
            max = arr[i];
    return max;
}

int main()
{
    int arr1[6] = {4, 6, 7, 8, 9, -1};
    int arr2[10] = {2, 3, 12, -4, 0, 6, 7, 13, 2, -3};
    int max1 = findMaxArray(arr1, 6);
    int max2 = findMaxArray(arr2, 10);
    if (max1 > max2)
        printf("The maximum is: %d
", max1);
    else
        printf("The maximum is: %d
", max2);
    return 0;
}
```
Calling a Function

• Function call: name of function followed by arguments

\[
\text{variable} = \text{function}_\text{name} ( \text{args}, \ldots );
\]

\[
z = \text{average}(x, y);
\]

\[
\text{average}(x, y); \quad //\text{don’t capture result}
\]

• () must be present even if function takes no parameters:

\[
\text{average}; \quad //\text{Wrong}
\]
Calling a Function

Function can be called as many times as needed

\[
\begin{align*}
z &= \text{average}(x, y); \\
m &= \text{average}(x, y); \\
n &= \text{average}(x, y); \\
p &= \text{average}(x, y);
\end{align*}
\]