Functions

What is a function?

Functions are self-contained program segments designed to perform a specific, well-defined task.

- All C programs have one or more functions
- The main() function is required
- Functions can accept parameters from the code that calls them
- Functions usually return a single value
- Functions help to organize a program into logical, manageable segments
Functions
Definitions

Syntax

- Data type of
- return
- expression

Parameter List
- (optional)

Header

Parameter List
- (optional)

Type
- identifier

Body

 declarations
- statements

return
- expression;

Example

Function Definitions: Syntax Examples

```c
int maximum(int x, int y)
{
    int z;
    z = (x >= y) ? x : y;
    return z;
}
```

Example – A more efficient version

```c
int maximum(int x, int y)
{
    return ((x >= y) ? x : y);
}
```

Functions
Function Definitions: Return Data Type

- A function's type must match the type of data in the return expression

- A function may have multiple return statements, but only one will be executed and they must all be of the same type

Example

Function Definitions: Return Data Type

```c
int bigger(int a, int b)
{
    if (a > b)
        return 1;
    else
        return 0;
}
```
### Functions

Function Definitions: Return Data Type

- The function type is **void** if:
  - The `return` statement has no expression
  - The `return` statement is not present at all
- This is sometimes called a **procedure function** since nothing is returned

#### Example

```c
void identifier(type1 arg1, ..., type_n arg_n)
{
    declarations
    statements
    return;  // may be omitted if nothing is being returned
}
```

### Functions

Function Definitions: Parameters

- A function's parameters are declared just like ordinary variables, but in a comma delimited list inside the parentheses
- The parameter names are only valid inside the function (local to the function)

#### Syntax

```c
type identifier(type1 arg1, ..., type_n arg_n)
{
    declarations
    statements
    return expression;
}
```

### Functions

Function Definitions: Parameters

- Parameter list may mix data types
  - `int foo(int x, float y, char z)`
- Parameters of the same type must be declared separately – in other words:
  - `int maximum(int x, y)` will **not** work
  - `int maximum(int x, int y)` is correct

#### Example

```c
int maximum(int x, int y)
{
    r = x > y ? x : y;
    return r;
}
```

### Functions

Function Definitions: Parameters

- If no parameters are required, use the keyword **void** in place of the parameter list when defining the function

#### Example

```c
type identifier(void)
{
    declarations
    statements
    return expression;
}
```
Functions

How to Call / Invoke a Function

Function Call Syntax

- No parameters and no return value
  \texttt{foo();}
- No parameters, but with a return value
  \texttt{x = foo(); \quad x (f_{oo})}
- With parameters, but no return value
  \texttt{foo(a, b);}
- With parameters and a return value
  \texttt{x = foo(a, b);}

Functions

Function Prototypes

- Just like variables, a function must be declared before it may be used
- Declaration must occur before \texttt{main()} or other functions that use it
- Declaration may take two forms:
  - The entire function definition
  - Just a function prototype – the function definition itself may then be placed anywhere in the program

Function Prototypes

- Function prototypes may be take on two different formats:
  - An exact copy of the function header:
    - Example – Function Prototype 1
      \begin{verbatim}
      int maximum(int x, int y)
      {
        return ((x >= y) \ ? \ x : y);
      }
      \end{verbatim}
  - Like the function header, but without the parameter names – only the types need be present for each parameter:
    - Example – Function Prototype 2
      \begin{verbatim}
      int maximum(int, int); \ NOT \ RECOMMENDED.
      \end{verbatim}

Functions

Declaration and Use: Example 1

\begin{verbatim}
int a = 5, b = 10, c; \GLOBAL \ VARIABLES \ (RECOMMENDED)
int maximum(int x, int y)
{
  return ((x >= y) \ ? \ x : y);
}

int main(void)
{
  c = maximum(a, b);
  printf("The max is \%d\n", c)
}
\end{verbatim}
Example 2

Declaration and Use: Example 2

```c
int a = 5, b = 10, c;
int maximum(int x, int y);
int main(void)
{
    c = maximum(a, b);
    printf("The max is %d\n", c);
}

int maximum(int x, int y)
{
    return ((x >= y) ? x : y);
}
```

Function is declared with prototype before use in main()
Function is defined after it is used in main()

Example

Passing Parameters by Value

```c
int a, b, c;
int foo(int x, int y)
{
    x = x + (x + y);
    return x;
}
int main(void)
{
    a = 5;
    b = 10;
    c = foo(a, b);
}
```

The value of a is copied into x.
The value of b is copied into y.
The function does not change the value of a or b.

```c
long int factorial(int n)
{
    if (n <= 1)
        return 1;
    else
        return n * factorial(n - 1);
}
```

Example: Factorials (5! = 5 * 4 * 3 * 2 * 1)

- Parameters passed to a function are passed by value.
- Values passed to a function are copied into the local parameter variables.
- The original variable that is passed to a function cannot be modified by the function since only a copy of its value was passed.

Recursion

A function can call itself repeatedly.
Useful for iterative computations (each action stated in terms of previous result).
Example: Factorials (5! = 5 * 4 * 3 * 2 * 1)
Functions

Evaluation of Recursive Functions

- Evaluation of 5!
  (based on code from previous slide)

<table>
<thead>
<tr>
<th>Recursive iterations of function</th>
<th>Partial results pushed on stack</th>
<th>Factorial term replaced with result of expression above</th>
<th>Result evaluated from TOS downward</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0] 1! = 1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>[1] 2! = 2 * 1!</td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>[2] 3! = 3 * 2!</td>
<td></td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>[3] 4! = 4 * 3!</td>
<td></td>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>[4] 5! = 5 * 4!</td>
<td></td>
<td>5</td>
<td>120</td>
</tr>
</tbody>
</table>

Conceptual evaluation of recursive function

Functions and Scope

Parameters

- A function’s parameters are local to the function – they have no meaning outside the function itself

- Parameter names may have the same identifier as a variable declared outside the function – the parameter names will take precedence inside the function

These are not the same n.

```c
int n;
long int factorial(int n){...}
```

Functions and Scope

Variables Declared Within a Function

- Variables declared within a function block are local to the function

```c
int x, y, z;

int foo(int n)
{
    int a;
    a += n;
    // The a refers to the a declared locally within the function body
    return a;
}
```

Functions and Scope

Variables Declared Within a Function

- Variables declared within a function block are not accessible outside the function

```c
int x;
int foo(int n)
{
    int a;
    return (a += n);
}

int main(void)
{
    x = foo(5);
    // This will generate an error. a may not be accessed outside of the function where it was declared.
    x = a;
}
```
Example

Functions and Scope

Global versus Local Variables

```
#include <stdio.h>

int x = 5;
int foo(int y)
{
    int z = 1;
    return (x + y + z);
}

int main(void)
{
    int a = 2;
    x = foo(a);
    a = foo(x);
}
```

- `x` can be seen by everybody
- `foo`'s local parameter is `y`
- `foo`'s local variable is `z`
- `foo` cannot see `main`'s `a`
- `foo` can see `x`
- `main`'s local variable is `a`
- `main` cannot see `foo`'s `y` or `z`
- `main` can see `x`

Example

```
int x = 5;
int foo(int y)
{
    int z = 1;
    return (x + y + z);
}

int main(void)
{
    int a = 2;
    x = foo(a);
    a = foo(x);
}
```

- Different functions may use the same parameter names
- The function will only use its own parameter by that name

Example

```
#define x 2

void test(void)
{
    #define x 5
    printf("%d\n", x);
}

void main(void)
{
    printf("%d\n", x);
    test();
}
```

- Running this code will result in the following output in the Uart1 IO window:
- `5`
- `5`

Why?
- Remember: `#define` is used by the preprocessor to do text substitution before the code is compiled.
Functions

Historical Note

• C originally defined functions like this:

```c
int maximum(int x, int y)
{
    return ((x >= y) ? x : y);
}
```

• Do not use the old method – use the new one only:

```c
int maximum(int x, int y)
{
    return ((x >= y) ? x : y);
}
```

Exercise 08

Functions

• Open the lab Project:

  On the class website
  Examples -> Lab8.zip

  Open MPLAB® X and select Open Project Icon (Ctrl + Shift + O)
  Open the Project listed above.
  If you already have a project open in MPLAB X, close it by “right clicking” on the open project and selecting “Close”

  Solution: Step 1

```c
#define MAX

int multiply_function(int x, int y);
//multiply_function() prototype
float divide_function(float x, float y);
//divide_function() prototype
```
Exercise 08

Solution: Step 2

```c
/*############################################################################ # STEP 2: Call the multiply_function() and divide_function(). # (a) Pass the variables intVariable1 and intVariable2 to the multiply_function(). # (b) Store the result of multiply_function() in the variable "product" # (c) Pass the variables floatVariable1 and floatVariable2 to the divide_function(). # (d) Store the result of divide_function() in the variable "quotient".

//Call multiply_function
product = multiply_function( intVariable1 , intVariable2 );

//Call divide_function
quotient = divide_function( floatVariable1 , floatVariable2 );

// intQuotient will be 0 since it is an integer
intQuotient = divide_function( floatVariable1 , floatVariable2 );
```

Solution: Steps 3 and 4

```c
/*############################################################################ # STEP 3: Write the function multiply_function(). Use the function prototype # you wrote in STEP 1 as the function header. In the body, all you need to do is return the product of the two input parameters (x * y)

//Function Header
int multiply_function( int x, int y)
{
    return (x * y);
    //Function Body

/*############################################################################ # STEP 4: Write the function divide_function(). Use the function prototype # you wrote in STEP 1 as the function header. In the body, all you need to do is return the quotient of the two input parameters (x / y)

//Function Header
float divide_function( float x, float y)
{
    return (x / y);
    //Function Body
```

Exercise 08

Conclusions

- Functions provide a way to modularize code
- Functions make code easier to maintain
- Functions promote code reuse

Questions?
```c
if (top == 0) {
    printf("Stack Underflow!
");
}
else {
    nth = top;
    top = top - 1;
    return 1;
}
```

```
if (top == 0) {
    printf("Stack Underflow!
");
}
else {
    nth = top;
    top = top - 1;
    return 1;
}
```