C Code Part 2

Objectives

- Understand arrays
- Understand functions
- Create and use multi-file projects

Arrays

How to Create an Array

Arrays are declared much like ordinary variables:

Syntax

```
type arrayName[size];
```

- `size` refers to the number of elements
- `size` must be a constant integer

Example

```
int a[10];    // An array that can hold 10 integers

char s[25];   // An array that can hold 25 characters
```

Arrays

Definition

Arrays are variables that can store many items of the same type. The individual items known as **elements**, are stored sequentially and are uniquely identified by the array **index** (sometimes called a **subscript**).

- **Arrays:**
  - **May contain any number of elements**
  - **Elements must be of the same type**
  - **The index is zero based**
  - **Array size (number of elements) must be specified at declaration**

Initialize Array

How to Initialize an Array at Declaration

Arrays may be initialized with a list when declared:

Syntax

```
type arrayName[size] = {item1,…,itemn};
```

- The items must all match the *type* of the array

Example

```
int a[5] = {10, 20, 30, 40, 50};

char b[5] = {'a', 'b', 'c', 'd', 'e'};
```

How to use an array

Arrays are accessed like variables, but with an index:

Syntax

```
arrayName[index]
```

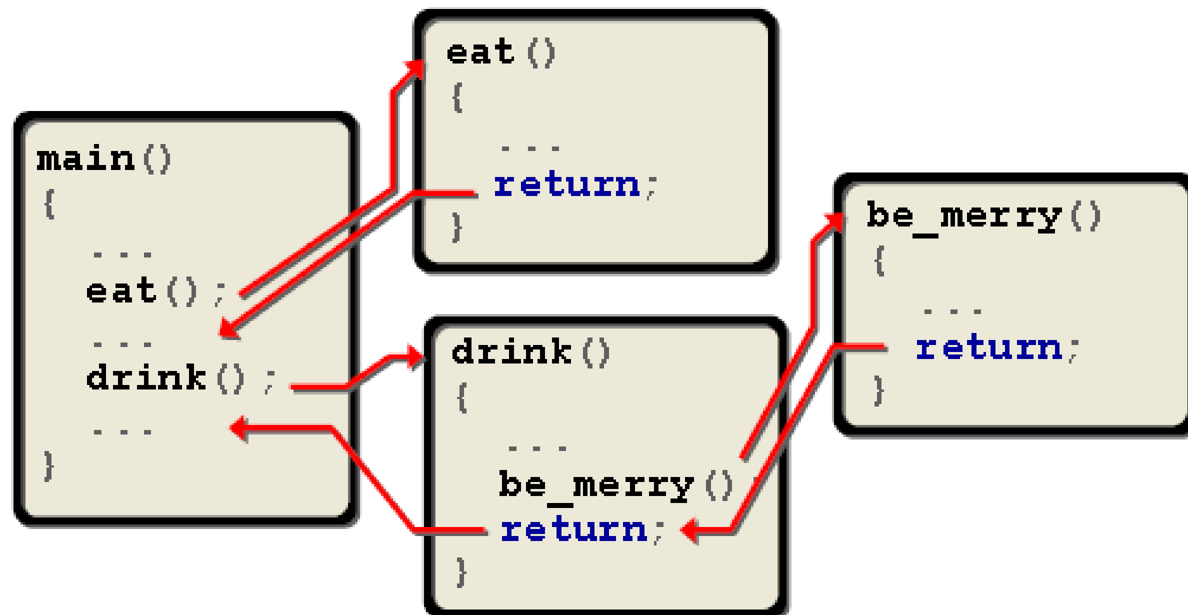
- *index* may be a variable or a constant
- The first element in the array has an index of 0
- C does not provide any bounds checking

Example

```
int i, a[10];   //An array that can hold 10 integers

for(i = 0; i < 10; i++) {
  a[i] = 0;     //Initialize all array elements to 0
}
a[4] = 42;      //Set fifth element to 42
```

Functions



Functions

What is a function?

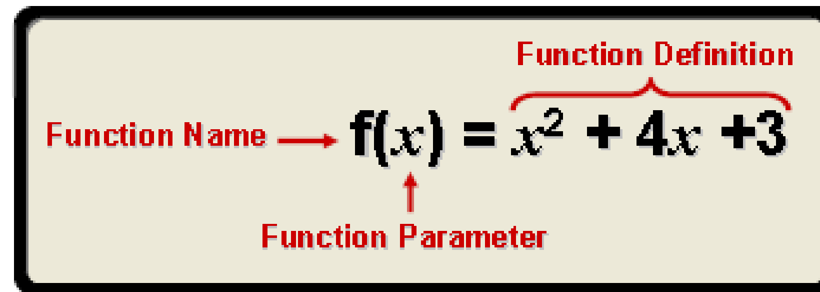
Definition

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

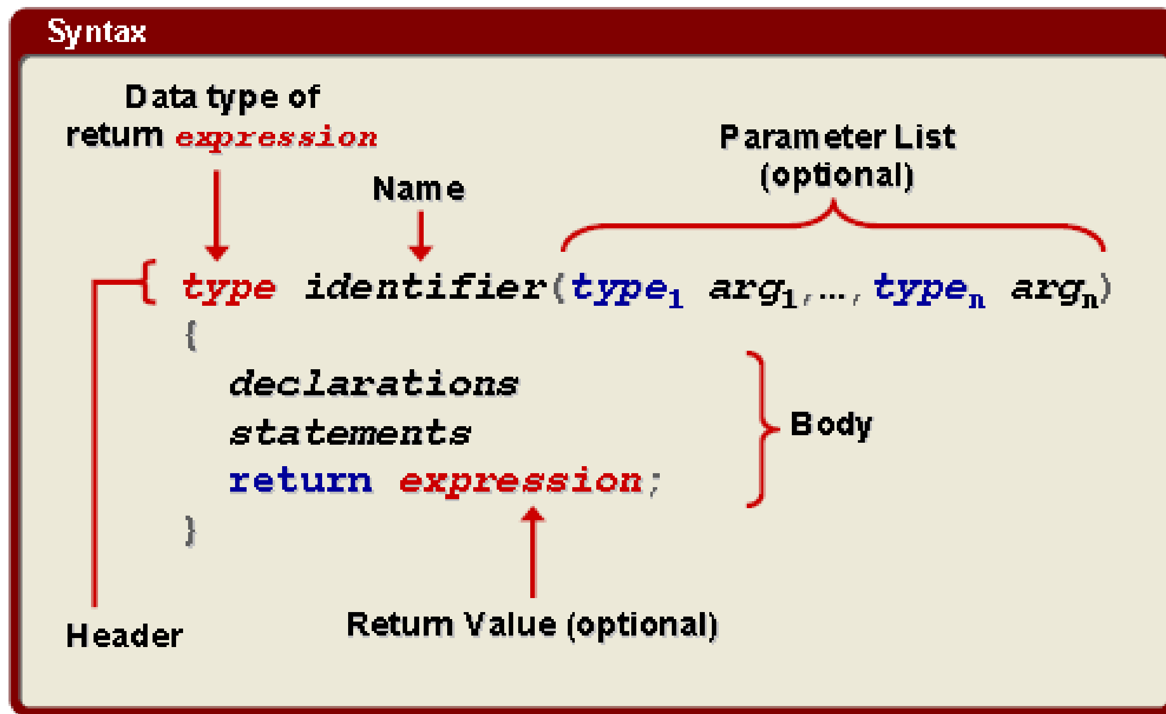
Remember algebra?

- Functions in C are conceptually like an algebraic function from math class…



- If you pass a value of 7 to the function: f(7), the value 7 gets "copied" into x and used everywhere that x exists within the function definition: $f(7) = 7^2 + 4*7 + 3 = 80$

Function Definition



Function Examples

Function Definitions: Syntax Examples

Example

```
int maximum(int x, int y)
{
  int z;

  z = (x >= y) ? x : y;
  return z;
}
```

Example – A more efficient version

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

Function Return type

Function Definitions: Return Data Type

Syntax

```
type identifier(type1 arg1,…, typen argn)
{
  declarations
  statements
  return expression;
}
```

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

no return 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

```
void identifier(type_1 arg_1, ..., type_n arg_n)
{
   declarations
   statements
   return;
}
```

`return;` may be omitted if nothing is being returned

Function Parameters

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

```
type identifier(type1 arg1,…,typen argn)
{
  declarations
  statements
  return expression;
}
```

Function Parameters: `type1 arg1,…,typen argn`

No Parameters

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

Example

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

How to invoke a function

How to Call / Invoke a Function

Function Call Syntax

- No parameters and no return value

```
foo();
```

- No parameters, but with a return value

```
x = foo();
```

- With parameters, but no return value

```
foo(a, b);
```

- With parameters and a return value

```
x = foo(a, b);
```

Function declaration

- Just like variables, a function must be declared before it may be used
- Declaration must occur before 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

Prototypes

- Function prototypes may take on two different formats:
  - An exact copy of the function header:

Example – Function Prototype 1

```
int maximum(int x, int y);
```

  - Like the function header, but without the parameter names – only the types need be present for each parameter:

Example – Function Prototype 2

```
int maximum(int, int);
```

Function Declare and Use before MAIN

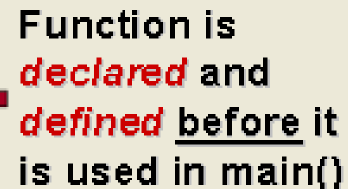
Declaration and Use: Example 1

Example 1

```
int a = 5, b = 10, c;

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)
}
```



Function declare and use
Body after main

Declaration and Use: Example 2

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()

Passing parameters

- 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

Passing values

Passing Parameters by Value

Example

```c
int a, b, c;

int foo(int x, int y)
{
  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.

Function variables and Scope

- 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.

```
int n;
long int factorial(int n){…}
```

Variable scope

Variables Declared Within a Function

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

Example

```
int x, y, z;

int foo(int n)
{
        int a;

        a += n;
}
```

The **n** refers to the function parameter **n**

The **a** refers to the **a** declared locally within the function body

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Global versus Local

Global versus Local Variables

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);
}
```

- x can see 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

Fundamentals of C

A Simple C Program

Example

Preprocessor Directives

Header File

```c
#include <stdio.h>

#define PI 3.14159

int main(void)
{
    float radius, area;

    //Calculate area of circle
    radius = 12.0;
    area = PI * radius * radius;
    printf("Area = %f", area);
}
```

Constant Declaration (Text Substitution Macro)

Variable Declarations

Comment

Function

Include Directive

- Three ways to use the `#include` directive:

Syntax

```
#include <file.h>
```

Look for file in the compiler search path
The compiler search path usually includes the compiler's directory and all of its subdirectories.
For example: C:\Program Files\Microchip\MPLAB C30*.*

```
#include "file.h"
```

Look for file in project directory only

```
#include "c:\MyProject\file.h"
```

Use specific path to find include file

Include Example

main.h Header File and main.c Source File

main.h

```c
unsigned int a;
unsigned int b;
unsigned int c;
```

The contents of main.h are *effectively* pasted into main.c starting at the `#include` directive's line

main.c

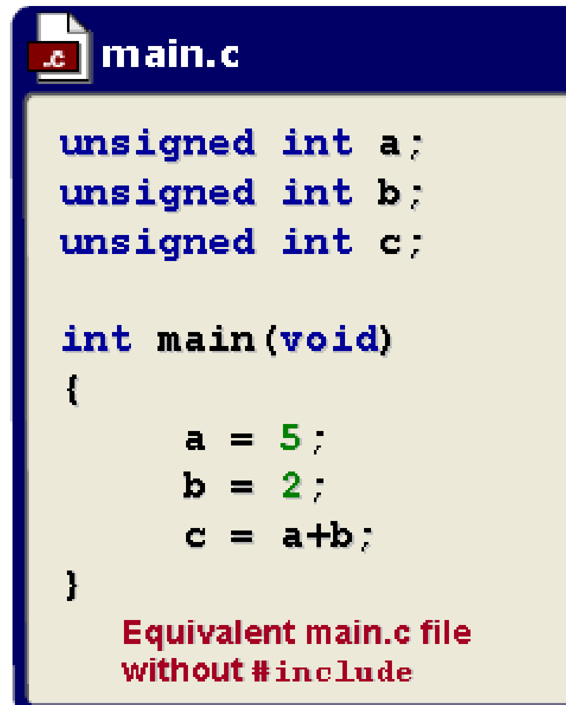
```c
#include <main.h>

int main(void)
{
        a = 5;
        b = 2;
        c = a+b;
}
```

Include equivalent

Equivalent main.c File

- **After the preprocessor runs, this is how the compiler sees the main.c file**
- **The contents of the header file aren't *actually* copied to your main source file, but it will behave *as if* they were copied**



Multi-project files

- Multi-file projects take the concept of functions further, by providing an additional level of modularization
- Globally declared variables and all normal functions are externally available if external declarations and function prototypes are available