Lecture 4
Functions
Arguments
Functions

• Functions (or subroutines) are used to encapsulate some computation that is going to be frequently repeated in a program.
  – Often, functions are used to simply clarify code even though the function may not be repeatedly called.
#include <stdio.h>

int power(int base, int n);

/* test power function */
main()
{
    int n;
    int base;
    int two, three;

    for (n=0; n < 10; ++n) {
        base = 2;
        two = power(base, n);
        base = 3;
        three = power(base, n);
        printf("%d %d %d\n", n, two, three);
    }
    return 1;
}

// int power(base, n)
// int base;
// int n;
int power(int base, int n)
{
    int i, p;
    p = 1;
    for (i=1; i <=n; ++i) {
        p = p*base;
    }
    base = 12;
    n =14;
    return p;
}
Function Definitions

• Function definitions
  – Have this form
    
    \[\text{return-type function-name(parameter declarations, if any)}\]
    
    \[
    \begin{align*}
    &\{ \\
    &\quad \text{declarations} \\
    &\quad \text{statements} \\
    &\}
    \end{align*}
    \]

  – can appear in any order
  – in one file or several
int power (int m, int n);

• This line, before main, is called a function prototype.
• int m and int n are called parameters
  – parameters are also called formal arguments
• The declared function must agree with this prototype
  – variable names do not have to agree and are not, in fact, required
  – variable names are used for documentation
power.c

```c
return 0;
```

- This line, at the end of main, returns a status to the invoking shell.
- This can be checked with the `$status` variable.
  - `echo $? after running the code.`
int power(int base, int n)

• int power states that this function return a value of type int. Why is this important?
• int base and int n are called *arguments*
  – *arguments* are also called *actual arguments*
    • Some compilers will refer to a mismatch with the prototype by stating that the *formal arguments* do not match the *actual arguments*.
  – Variables base and n are local to the power function.
    • Variables of the same name, elsewhere in the program are different variables
    • Anything changed in these variables is only changed in this function.
  – The variable names may be different than the prototype.
int i, p;

- These variables, declared within the function, are local to the function.
  - No other code has access to them.
  - Other variables of the same name, elsewhere in the program, are different variables.
return p;

• This line, at the end of the function, returns a value of type int to the caller.
• The value is the value stored in p.
• The return type is specified by the function declaration and must match.
• functions do not need to return a value.
  – return;
  – such functions should have the return value void
    • void function_name(arguments)
Early versions of C

- Function prototypes did not include formal argument lists
  ```c
  int power();
  ```
- Function declarations did not allow types in the argument list
  ```c
  int power(base, n)
  int base, n;
  ```
Arguments – Call by Value

• Call by Value means that the copies of the values of the arguments are passed from the caller to the function rather than the actual variables themselves.
  – There is a way to pass the variables themselves
    • Call by Reference
    • We will examine it later.

• Changing the value of a passed argument has no effect on the value of the variable in the calling routine.
Arrays and Call by Value

- Arrays work slightly differently. The array variable is really an address to a memory space containing the array.
- Any change in the array is really a change in the memory space referenced by the array.
- As the copies of the array are pointing to the same place in memory, any change to an array IS reflected in the calling function.
Handout #2 – Trace the following program

```c
int value (int p, int q);

main () {
    int p, q;
    q = 2;
    p = 4;
    q = value(p,q);
    q = p;
}

int value (int p, int q) {
    p = 7;
    q = p;
    return q;
}
```

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>
int value (int p, int q);

main () {
    int p, q;
    q = 2;
    p = 4;
    q = value(p,q);
    q = p;
}

int value (int a, int b) {
    a = 7;
    b = a;
    return b;
}