Lesson 6

Functions

Class Objectives

In this lesson, you will learn about:

- Calling standard functions
- Function parameters
- Function return types
- Returning values from functions
- Writing your own functions

Important Notes:

- **You should type in all the programs in this handout, and run them more than once with different data**
- **You should read, and understand everything in this handout, the material in it forms the basis of the quizzes**
- **If you don't understand something; ask me to explain.**

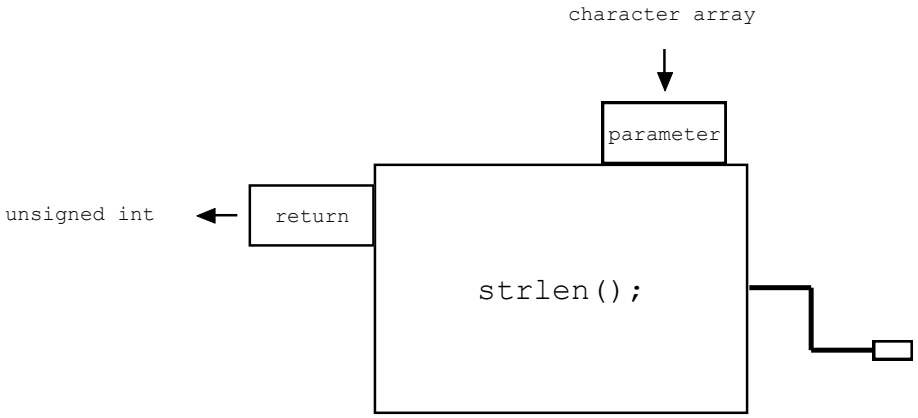
What are Functions?

- Functions can be accurately described as little programs that run within programs.
- They are distinct units that require data values to be supplied to them (parameters), from the main program, and give back (return) to the main program other data values.
- A function may have many parameters, but will *return* only one value.
- All programming languages have a set of standard functions, and we have already used some: `strlen()`, `strcmp()`, and `strcpy()`.

Function (`strlen()`)

The standard function `strlen()` has one parameter. The type of that parameter is a character array. It's return type is an unsigned integer.

A pictorial representation of `strlen()` is below:



If the function were a mechanical device, you would place the string in the parameter hopper on the top, crank the handle, and the number of character in the string would pop out the return pipe.

On the next page is a simple program that demonstrates a call of the function `strlen()`.

Program (`string1.c`)

```
/*
  Program string1.c
  written by: Joe Dorward
  Date: 04/18/00

  This program demonstrates a call of the standard function strlen();
*/

#include <stdio.h>
#include <string.h>

void main(void)
{
unsigned int string_length;

char the_string[25];

  printf("\n Please enter a string: ");
  scanf("%s",the_string);

  string_length = strlen(the_string);   /* function call */

  printf("\n The length of: %s,",the_string);
  printf(" is: %d characters.",string_length);
}
```

When you run this program, type in the string “hello”, at the prompt. The number 5 will be printed to the screen.

Defining functions

All functions need to be defined, this means:

- Its return type needs to be stated
- The name of the function needs to be stated
- The type of all parameters need to be stated

If we had written the function `strlen()`, we might have defined it as:

```
unsigned int strlen(char a_string[50]);
```

`unsigned int` tells us that we expect a value of that type to be returned.
`strlen` is the name of the function, and it’s the name we use to call the function.
`char a_string[50]` is the type, and name of the variable used inside the function.

NOTE:

Because `strlen()` returns an unsigned integer, the variable that takes its value must be an unsigned integer.

Calling other standard C functions

The square root function `sqrt()`, is one of the standard functions defined in the header file `math.h`, others include: `sin()`, `cos()`, and `tan()`. The square root of a number, is the number, when multiplied by itself, results in the first number.

For example:

- 2 is the square root of 4
- 3 is the square root of 9
- 4 is the square root of 16
- 5 is the square root of 25

To use a function, you must know:

- The return type of the function
- The number of parameters
- The type(s) of the parameters
- The type order of the parameters (Are there 2 integers, then a float? Or vice-versa?)

To know all that, you need to look at the definitions contained in the appropriate header file. An extract of `math.h`, shows:

```
int          abs   (int x);

double       cos   (double x);
double       sin   (double x);
double       tan   (double x);
double       sqrt  (double x);

double       pow   (double x, double y);
```

This shows us that:

`abs()`, takes a single `int` parameter, and returns a value of type `int`

`cos()`, `sin()`, `tan()`, `sqrt()`, each take a single `double` parameter, and return a value of type `double`

`pow()`, takes a pair of `double` parameters, and returns a value of type `double`

With that knowledge, we can *call* these functions in programs.

Program (sqrt.c)

```
/*
  Program "sqrt.c"
  written by: Joe Dorward
  Date: 04/18/00

  This program demonstrates a call of the standard function sqrt()
*/

#include <stdio.h>
#include <math.h>

void main(void)
{
double the_number,
       the_answer;

  printf("\n Please enter an real number: ");
  scanf("%lf",&the_number);

  the_answer = sqrt(the_number);   /* function call */

  printf("\n The square root of: %lf,",the_number);
  printf(" is: %lf",the_answer);
}
```

Writing your own functions

Function (`add_two()`)

Define the function prototype:

```
int add_two(int,int);
```

The return type of this function is `int`, and it takes in two `int` parameters.

Declare the function:

```
int add_two(int first, int second);
{
  return(first + second);
}
```

NOTE:

- The integer variables used inside the function are `first`, `second`
- The calculation is done within the `return()` statement. It is this return statement that puts the result of the calculation back out to the main program.
- The function declaration goes after the closing `}` of the main program.
- The function prototype goes after the `#include` statements

Program (`fun1.c`)

```
/*
  Program "fun1.c"
  written by: Joe Dorward
  Date: 04/20/00

  This program demonstrates writing, and calling a function
*/

#include <stdio.h>

int add_two(int,int);   /* function prototype */

void main(void)
{
int number_one,
    number_two,
    the_answer;

  printf("\n Please enter a number: ");
  scanf("%d",&number_one);

  printf("\n Please enter a number: ");
  scanf("%d",&number_two);

  the_answer = add_two(number_one,number_two);   /* function call */

  printf("\n Adding %d and %d",number_one,number_two);
  printf(" = %d \n",the_answer);
}
/* ================================================================== */
int add_two(int first,int second)   /* function definition */
{
  return(first + second);
}
/* ================================================================== */
```

NOTE:

You will notice that function `add_two()` is called with the parameter variables `number_one`, and `number_two`, while the variable names used inside the function are `first`, and `second`.

The variable names inside a function, have nothing to do with the variable names used to call the function.

The only connection between the external, and internal variables it that their types must be the same.

Remember that we have no idea what the internal names are of the standard functions, but we are still able to call them.

Function (`odd_even1()`)

This function will tell the program if the integer passed to it is an even number.

If the parameter is even, it will return a 1, if the parameter is odd, it will return 0.

Define the function prototype:

```
int is_it_even1(int);
```

The return type of this function is `int`, and it takes in one `int` parameter.

Declare the function:

```
int is_it_even1(int the_number);
{
  if (the_number % 2 == 0)   // it's an even number
  {
    return(1);
  }
  else   // must be an odd number
  {
    return(0);
  }
}
```

Program (even1.c)

```
/*
  Program "even1.c"
  written by: Joe Dorward
  Date: 04/20/00

  This program demonstrates writing, and calling a function that
  decides if a number is odd or even.
*/

#include <stdio.h>

int is_it_even1(int);   /* function prototype */

void main(void)
{
int number_one;

  printf("\n Please enter a number: ");
  scanf("%d",&number_one);


  if (is_it_even1(number_one) == 1)   /* function call */
  {
    printf("\n The integer %d is even. \n",number_one);
  }
  else
  {
    printf("\n The integer %d is odd. \n",number_one);e
  }
}
/* ========================================================================= */
int is_it_even1(int the_number)   /* function definition */
{
  if (the_number % 2 == 0)   /* its an even number */
  {
    return(1);
  }
  else   /* must be an odd number */
  {
    return(0);
  }
}
/* ========================================================================= */
```

Function (`is_lowercase()`)

This function will tell the program if the character passed to it is a lowercase letter.

Define the function prototype:

```
int is_lowercase(char);
```

The return type of this function is `int`, and it takes in one `char` parameter.

Declare the function:

```
int is_lowercase(char character)   /* function definition */
{
  if ( (character >= 'a') && (character <= 'z') )   /* true if lowercase */
  {
    return(1);
  }
  else
  {
    return(0);
  }
}
```

Program (is_lower2.c)

```
/*
  Program "is_lower2.c"
  written by: Joe Dorward
  Date: 04/22/00

  The function in this program, returns a 1 if its character parameter
  is a lowercase letter, and 0 otherwise.
*/

#include <stdio.h>

int is_lowercase(char);   /* function prototype */

void main(void)
{
char the_character;

/* ================================================================= */

  printf("\n Please enter a lowercase letter: ");
  scanf("%c",&the_character);


  if (is_lowercase(the_character) == 1)   /* function call in condition */
  {
    printf("\n The character: \'%c\' ",the_character);
    printf("is a lowercase letter.");
  }
  else
  {
    printf("\n The character: \'%c\' ",the_character);
    printf("is not a lowercase letter.");
  }
}
/* ================================================================= */
int is_lowercase(char character)   /* function definition */
{
  if ( (character >= 'a') && (character <= 'z') )   /* true if lowercase */
  {
    return(1);
  }
  else
  {
    return(0);
  }
}
/* ================================================================= */
```

Program (to_upper2.c)

```
/*
  Program "to_upper2.c"
  written by: Joe Dorward
  Date: 04/22/00

  The function in this program, returns an uppercase letter,
  of a lower case letter.
*/

#include <stdio.h>

char lower_to_upper(char);   /* function prototype */

void main(void)
{
char the_character;

  printf("\n Please enter a lowercase letter: ");
  scanf("%c",&the_character);


  printf("\n The uppercase version of: \'%c\' ",the_character);
  printf("is: \'%c\' ",lower_to_upper(the_character));   /* function call */
}
/* ================================================================== */
char lower_to_upper(char character)   /* function definition */
{
  return(character - 32);
}
/* ================================================================== */
```