FIT ICT Software Development

Lecture 8

Employing Functions

Lecturer : Charles Tyner

FIT ICT Software Development

Declaring Functions

Most C programs contain custom functions which can be called as required

Function block contains one or more statements that are executed when function is called

On completion of function, program flow returns to statement immediately following function call

Must be declared before they can be called using a *function prototype*

Data type of return value, or keyword ***void*** if no value is returned, must be included in prototype

FIT ICT Software Development

Testing Expressions

Sometimes need to evaluate multiple expressions

Can be achieved in two ways:

1. Logical *&&* AND operator ensures statements only executed when both expressions found to be true

Syntax

If((test-expression_1)&&(test-expression_2))
{statements-to-execute

FIT ICT Software Development

Declaring Functions

1. Begin a new program with a preprocessor instruction to include standard input/output library:

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

2. Before the ***main()*** function, declare three custom function prototypes:

```
void first();
int square5();
int cube5();
```

3. Add main function that declares an integer variable:

```
int main() {
    int num;
}
```

FIT ICT Software Development

Declaring Functions

4. After the main function, define the custom functions

```
void first()
{
 printf(“Hello from the first function\n”);
}
int square5()
{
 int square = 5 * 5;
 return square;
}

int cube5()
{
 int cube = ( 5 * 5) * 5;
 return cube;
}
```

FIT ICT Software Development

Declaring Functions

5. In the main function, insert calls to custom functions

```
first();
num = square5();
printf("5x5= %d\n", num);
printf("5x5x5= %d\n", cube5());
```

6. Return zero. Save, compile and execute. Study output from custom functions. Note the two different ways the functions are invoked. In one, the result is assigned to a declared variable of the appropriate type; in the other, the result is displayed as output using a specifier.

FIT ICT Software Development

Function Arguments

Also known as ***formal parameters*** of a function

Function prototype must include name and data type of each argument

Arguments are passed by value, so function operates on local copy, not original value

Multiple formal parameters can be specified for a function, but must be separated by a comma

May be of different data types

Syntax:

```
void  action(char c, int i, float f, double d);
```

FIT ICT Software Development

Function Arguments

1. Begin a new program with a preprocessor instruction to include standard input/output library:

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

2. Before the *main()* function, declare three custom function prototypes with one argument to be passed:

```
void display(char str[]);
int square(int x);
int cube(int y);
```

3. Add main function that declares an integer variable and a character array variable initialised with a string:

```
int main() {
    int num;
    char msg[50] = “String to be passed to a function” ;
}
```

FIT ICT Software Development

Function Arguments

4. After the main function, define the custom functions

```
void display(char str[])
{
  printf(“%s\n”, str);
}
int square(int x)
{
 return (x * x );
}

int cube(int y)
{
 return (y * y) * y;
}
```

FIT ICT Software Development

Function Arguments

5. In the main function, insert calls to custom functions

```
display(msg);
num = square(4);
printf("4x4= %d\n", num);
printf("4x4x4= %d\n", cube(4));
```

6. Return zero. Save, compile and execute. Study output from custom functions.

FIT ICT Software Development

Recursive Functions

- Custom functions can call other custom functions as required
- Can also call themselves – ***recursive functions***
- Recursive functions not necessarily most efficient – loops can be and often are better
- Must modify a tested expression to avoid continuous execution

FIT ICT Software Development

Recursive Functions

1. Begin a new program with a preprocessor instruction to include standard input/output library:

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

2. Declare a custom function prototype with one argument to be passed:

```
void count_down_from(int num);
```

3. Add main function that declares an integer variable:

```
int main() {
    int start;
}
```

FIT ICT Software Development

Recursive Functions

4. In main function, add statement prompting user for integer , which will be assigned to a variable:

```
printf(“Enter a positive integer to count down from”);
scanf(“%d” , &start);
```

5. Now add a call to the custom function, passing the value entered by the user as a formal parameter:

```
count_down_from (start);
```

6. Add a statement to output a message when control returns from the custom function:

```
printf(“Lift off!\n”);
```

FIT ICT Software Development

Recursive Functions

7. At end of main block, return a zero value as required by main function declaration:

```
return  0;
```

8. Now add a custom function, beginning by outputting the argument value passed during the function call:

```
void count_down_from (int num)
{
printf(“%d\n”);
}
```

FIT ICT Software Development

Recursive Functions

9. Decrement the value passed as a formal parameter. This allows the processing to end at some point:

```
--num;
```

10. Now add a a test condition to return control to the main (calling) function if the value is now below zero, or pass it as a formal parameter in a recursive call to itself:

```
If (num < 0) return;
else
count_down_from (num);
```

11. Save, compile, run, study

FIT ICT Software Development

Placing Functions In Headers

- Can be sensible to collect frequently used functions in a custom header file
- Like standard header files, should have extension *.h*
- Made available to program by adding ***#include*** preprocessor directive at start of file containing main function
- Unlike standard header files, enclosed within <> angled brackets, custom header files are enclosed within double quotes

FIT ICT Software Development

Placing Functions In Headers

1. Create a custom header file named “utils.h” containing definition of a single function. Must be saved in same directory as file with main function:

```
int square( int num )
{
return ( num * num );
}
```

2. Begin a new program with a preprocessor instruction to include standard input/output library and the custom header file:

```
#include <stdio.h>
#include “utils.h”
```

FIT ICT Software Development

Placing Functions In Headers

3. Declare a custom function prototype with no arguments:

```
void getnum();
```

4. Add main function to call the other function and then return a zero

```
int main()
{
    getnum();
    return 0;
}
```

FIT ICT Software Development

Placing Functions In Headers

5. After main function, add second (custom) function definition bydeclaring two variables:

```
void getnum()
{
   int num ;
   char again ;
}
```

6. Add statements to request user input and assign the value to a variable:

```
printf("Enter an integer to be squared");
scanf(" %d" , &num);
```

FIT ICT Software Development

Placing Functions In Headers

7. Output result by calling function in custom header file, passing input value as its argument

```
printf(“%d squared is %d\n”, num , square(num));
```

8. Add statement to request further user input and assign the value to a variable:

```
printf(“Square another number? Y or N”);
scanf(“%1s” , &again);
```

Note use of %1s format specifier, which we haven’t used before. This reads the next character entered by the user

FIT ICT Software Development

Placing Functions In Headers

9. Add conditional test to either recursively call the function or return control to the calling function

```
if( (again == 'Y') || (again == 'y') ) getnum() ;
else return;
```

User is allowed to input upper or lower case *y*, but any other keypress will end execution

10. Save, compile, execute and study

FIT ICT Software Development

Static Functions

Static functions can only be accessed from within the files in which they are created

Recommended in larger programs using several .c files

Safeguards against accidental misuse of functions, or compilation failure because of identically named functions in different files

In example following, the custom functions are in a file other than that containing the main function, and so can’t be accessed directly from **main()**

FIT ICT Software Development

Static Functions

1. Begin a new program with a preprocessor instruction to include standard input/output library:

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

2. Declare a custom function prototype with no arguments:

```
void menu();
```

3. Add main function to call the custom function and then return a zero:

```
int main() {
  menu();
  return 0;
}
```

FIT ICT Software Development

Static Functions

4. After the main function block, define the prototyped custom function to pass a menu option as an argument in a call to another function:

```
void menu()
{
  int option ;
  printf(“\n\tWhat would you like to do?”);
  printf(“\n\t1. Square a number”);
  printf(“\n\t2. Multiply two numbers”);
  printf(“\n\t3. Exit”);
  scanf(“%d” , &option);
  action(option);
}
```

FIT ICT Software Development

Static Functions

5. Begin a second program with a preprocessor instruction to include standard input/output library:

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

6. Define two simple static functions. Note that no prototype is needed as these are outside the file containing the main function:

```
static square( int a ) { return (a* a) ; }
static multiply( int a, int b ) { return (a* b) ; }
```

FIT ICT Software Development

Static Functions

7. Now define the function that receives the call, and the menu option as an argument, from the file containing the main function:

```
void  action( int option)
{
   int n1, n2;
   if (option == 1)
   {
    printf("Enter an integer to be squared: ");
    scanf(" %d", &n1);
    printf(" %d x %d = %d\n", n1, n1, square(n1));
   }
```

FIT ICT Software Development

Static Functions

7. continued:

```
    else if (option == 2)
    {
      printf("Enter two integers to multiply ");
      printf("separated by a space");
      scanf("%d", &n1); scanf("%d", &n2);
      printf("%d x %d = %d\n", n1, n2, multiply(n1,
n2));
     }
     else return;
}
```

FIT ICT Software Development

Static Functions

8. Save the two files, the first as menu.c and the second as action.c. To compile multiple files, simply specify all source files and a single output filename using the following syntax:

```
gcc menu.c action.c –o menu.exe
```

9. Execute and examine the output returned from the static functions.