C Programming Basics

SDS 322/329

November 3, 2015

Email any questions to:
rauta@tacc.utexas.edu

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

Recap

- Arrays
- Structures
- Mid-Term
  - All the material was covered in the class and was from the slides/code-samples provided to you

Overview of the Course

- Writing a Basic C Program
- Understanding Errors
- Comments Keywords, Identifiers, Variables
- Operators
- Standard Input and Output (Basic)
- Control Structures
- Standard Input and Output
- Arrays, Structures
- Functions in C
- Pointers
- Working with Files

All the concepts will be accompanied with examples.

C Language Functions

- Functions are self-contained blocks of statements that perform a specific task

- Written once and can be used multiple times
  - Promote code reuse
  - Make code maintenance easy

- Two steps involved
  - Write the function
    - Function definition
    - Function declaration or prototype
  - Invoke or call the function

- Two types of functions
  - Standard or library or built-in
  - User-Defined

Standard Functions

- These functions are provided to the user in library files
- In order to use the functions, the user should include the appropriate library files containing the function definition
- Example
  - `scanf`
  - `printf`
  - `gets`
  - `puts`
  - `strcpy`

User-Defined Functions: see myFunction.c

Return type of function

Name of function

Arguments if any, none here

```
void add(){
   int a, b, c;
   printf("\n Enter Any 2 Numbers : ");
   fflush(stdout);
   scanf("%d %d",&a,&b);
   c = a + b;
   printf("\n Addition is : %d",c);
}
```

Body of function

Note: The function does not return any value and hence its return type is void and there is no return statement at the end of the function. Contrast this to the definition of function main.

User-Defined Functions: myFunction.c

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

Defining the function add

```
void add(){
   int a, b, c;
   printf("\n Enter Any 2 Numbers : ");
   fflush(stdout);
   scanf("%d %d",&a,&b);
   c = a + b;
   printf("\n Addition is : %d",c);
}
int main(){
   add();
   add();
   return 0;
}
```

Invoking the function add twice from function main

Function Prototype: myFctPrototype.c

```
#include <stdio.h>

void add();

int main(){
 add();
 return 0;
}

void add(){
int a, b, c;
printf("\n Enter Any 2 Numbers : ");
fflush(stdout);
scanf("%d %d",&a,&b);
c = a + b;
printf("\n Addition is : %d",c);
}
```

Function Prototype or Declaration: useful when the function is invoked before its definition is provided

Invoking the function add

Defining the function add that does not return a value - note void

Sending Input Values To Functions

- Determine the number of values to be sent to the function
- Determine the data type of the values that needs to be sent
- Declare variables having the determined data types as an argument to the function
- Use the values in the function
- Prototype the function if its definition is not going to be available before the place from where it is invoked
- Send the correct values when the function is invoked

Passing Values to Functions: passValue1.c

```
#include <stdio.h>
void add(int d, int e){ <-- Formal Parameters: d, e
  int c;
  c = d + e;
  printf("\n Addition is : %d",c);
}
int main(){
  int a, b;
  printf("\n Enter Any 2 Numbers : ");
  fflush(stdout);
  scanf("%d %d",&a,&b);
  add(1, 2); <-- Actual Parameters: 1, 2
  add(a, b); <-- Actual Parameters: a, b
  return 0;
}
```

Note: The variables used as formal and actual parameters can have different names.

Command-Line Arguments

- For passing arguments to the program through command-line, you will need a different signature of the main function
  - This is another way to write a program which can accept different values for a variable – you do not need to hard-code the values in the program

- The function main has two signatures as defined by the C programming standard

```
int main()
```

```
int main(int argc, char *argv[])
```

This signature is used for passing command-line arguments to the program

Passing Values to Functions: passValue2.c

```
#include <stdio.h>
#include <stdlib.h>
void add(int a, int b){
 //same code as in the previous slide
}
int main(int argc, char *argv[]){
 int a, b;
 if ( argc != 3 ){
  printf("\nInsufficient num. of arguments.\n");
  printf( "\nUsage:%s <firstNum> <secondNum>", argv[0]);
 }else{
    a = atoi(argv[1]);
    b = atoi(argv[2]);
    add(a, b);
 }
 return 0;
}
```

Code Snippet From passValue2.c

Notice that main has two arguments

```
int main(int argc, char *argv[]){
 int a, b;
 if ( argc != 3 ){
  printf("\nInsufficient num. of arguments.\n");
  printf( "\nUsage:%s <firstNum> <secondNum>", argv[0]);
 }else{
    a = atoi(argv[1]);
    b = atoi(argv[2]);
    add(a, b);
 }
 return 0;
}
```

argc is the argument count

argv[1] holds the first number typed in at the command-line. Notice the atoi function.

Returning Values from Functions: passValue4.c

```
#include <stdio.h>
int add(int a, int b){ <-- Notice the return type
   int c;
   c = a + b; a=c; b=c;
   printf("\n Addition is : %d",c);
   return c; <-- Return value: c
}
int main(){
   int a, b, c;
   printf("\n Enter Any 2 Numbers : ");
   scanf("%d %d",&a,&b);
   printf("a is: %d, b is: %d\n", a, b);
   c = add(a, b); <--- Value returned from add stored in c
   printf("a is: %d, b is: %d\n", a, b);
   return 0;
}
```

Returning Values from Functions: passValue4.c

- Output:

```
Enter Any 2 Numbers : 5 6
a is: 5, b is: 6
Addition is : 11
a is: 5, b is: 6
```

Note: the values of `a` and `b` remained the same when accessed from function main. More about functions on later slides

In-Class Exercise

- Modify the matrix-multiplication program to incorporate a user-defined function of your choice. For example,
  - you could write a `printMatrix` function that can print the contents of a two-dimensional matrix of type integer, and having 2 rows, and 2 columns
    - Can you modify the program further such that it accepts a matrix with any number of rows and columns?

References

- C Programming Language, Brian Kernighan and Dennis Ritchie
- Let Us C, Yashavant Kanetkar
- C for Dummies, Dan Gookin
- http://cplusplus.com