C Programming Basics – Part 2

Ritu Arora
Texas Advanced Computing Center
June 25, 2013

Email: rauta@tacc.utexas.edu
Overview of Content

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

All the concepts are accompanied by 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
#include <stdio.h>

--- Defining the function add ---

```c
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(); // Invoking the function add twice from function main
    add();
    return 0;
}
```
Function Prototype: myFctPrototype.c

```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

--- Invoking the function add
Categories of Functions

• Functions that take no input, and return no output

• Functions that take input and use it but return no output

• Functions that take input and return output

• Functions that take no input but return output
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
#include <stdio.h>

void add(int a, int b){
    int c;
    c = a + b;
    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(a, b);
    return 0;
}

Note: The variables used as formal and actual parameters can have different names.
#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;
}
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;
}
Returning Values from Functions: passValue4.c

```c
#include <stdio.h>

int add(int a, int b){
    int c;
    c = a + b; a=c; b=c;
    printf("Addition is : %d",c);
    return c;
}

int main(){
    int a, b, c;
    printf("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;
}
```

Return value:
c

Value returned from add stored in c

Notice the return type

12
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
Overview of Content

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

All the concepts are accompanied by examples.
Arrays

• An array allows you to store many different values of same data type in a single unit

• Arrays are declared just like other variables, though the variable name ends with a set of square brackets

  – `char myName[50];`  
  – `int myVector[3];`
  – `int myMatrix[3][3];`

←---- You have seen this before
Arrays Example: arrayExample.c

```c
#include <stdio.h>

int main(){
    int i;
    int age[4];
    age[0]=23; \textbf{Notice that count begins at 0}
    age[1]=34;
    age[2]=65;
    age[3]=74;
    for(i=0; i<4; i++){
        printf("age[%d]: %d\n", i, age[i]);
    }
    return 0;
}
```

Output:

- age[0]: 23
- age[1]: 34
- age[2]: 65
- age[3]: 74
Structures

• Multiple variables can be combined into a single package called structure
• Members of the structure variable need not be of the same type
• They can be used to do database work in C! Example:

```c
struct sample{
    int a;
    char b;
}
struct sample mySample;
```

• `typedef` is the keyword that can be used to simplify the usage of `struct`

```c
typedef struct sample newType;
```
## Structure Example: `structExample.c`

```c
#include <stdio.h>

typedef struct point{
  double x;
  double y;
}point;

int main(){
  point myPoint;
  myPoint.x = 12.2;
  myPoint.y = 13.3;
  printf("X is %lf and Y is %lf\n",myPoint.x, myPoint.y);
  return 0;
}
```

Notice the “.” operator
Overview of Content

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

All the concepts are accompanied by examples.
Pointers

• A pointer is a variable that stores an address in memory - address of another variable

• For instance, the value of a pointer may be 42435. This number is an address in the computer's memory which is the start of some data

• We can dereference the pointer to look at or change the data

• Like variables, you have to declare pointers before you use them

• The data type specified with pointer declaration is the data type of the variable the pointer will point to
Revisiting Variable Declaration

• Consider the declaration

```c
int i = 3;
```

• This declaration tells the C compiler to:
  – Reserve space in memory to hold the integer value
  – Associate the name `i` with this memory location
  – Store the value `3` at this location

```
i ------------ Location name

3 ------------ Value at location

6485 ------------ Location number
(Address)
```


`
#include <stdio.h>

int main() {
    int i = 3;
    printf("\nAddress of i = %u", &i);
    printf("\nValue of i = %d", i);
    printf("\nValue of i = %d", *(&i));
    return 0;
}
`

Output:
Address of i = 2293532
Value of i = 3
Value of i = 3

Note:

`&i` returns the address of variable `i`

`*(&i)` returns the value at address of `i`
Pointer Expressions

• In the previous example, the expression `&i` returns the address of `i`.
• This address can be collected in a variable as
  
  \[ j = \&i; \]
  
  • \( j \) is a variable which contains the address of another variable and is declared as `int *j;`

  \[
  i \leftarrow \text{Location name} \rightarrow j
  \]

  \[
  3 \leftarrow \text{Value at location} \rightarrow 6485
  \]

  \[
  6485 \leftarrow \text{Location number} \rightarrow 3276
  \]
  
  (Address)
#include <stdio.h>

int main(){
    int i=3;
    int *j;
    j = &i;

    printf("\nAddress of i = %u", &i);
    printf("\nAddress of i = %u", j);
    printf("\nAddress of j = %u", &j);
    printf("\nValue of j = %u", j);
    printf("\nValue of i = %d", i);
    printf("\nValue of i = %d", *(&i));
    printf("\nValue of i = %d", *j);
    return 0;
}
Key Concepts Related to Pointers

- Declaring a pointer

```c
int *myIntPtr;
int* myIntPtr;
```

- Getting the address of a variable

```c
int age = 3;
myIntPtr = &age;
```

- Dereferencing a pointer

```c
*myIntPtr = 5;
```

Note: We just changed the value of age!
#include <stdio.h>

int main(){
    int myValue;
    int *myPtr;
    myValue = 15;
    myPtr = &myValue;
    printf("myValue is equal to : %d\n", myValue);
    *myPtr = 25;
    printf("myValue is equal to : %d\n", myValue);
}

Output:
myValue is equal to : 15
myValue is equal to : 25
Pointers and Arrays

• The square-bracket array notation is a short cut to prevent you from having to do pointer arithmetic

```c
char array[5];
array[2] = 12;

array is a pointer to array[0]

array[2] = 12; is therefore equivalent to
*(array+2) = 12;
```
#include <stdio.h>

void addUpdate(int *a, int *b){
    int c;
    c = *a + *b;
    printf("Addition is : %d\n",c);
    *a = c;
    *b = c;
}

int main(){
    int a, b;
    printf("Enter Any 2 Numbers : ");
    scanf("%d %d", &a, &b);
    printf("a is: %d, b is: %d\n", a, b);
    addUpdate(&a, &b);
    printf("a is: %d, b is: %d\n", a, b);
    return 0;
}

Note: The values of a and b changed in addUpdate function.
Output of passValue3.c

• Output:
Enter Any 2 Numbers : 2 8
a is: 2, b is: 8
Addition is : 10
a is: 10, b is: 10
Dynamic Memory Allocation

- Dynamic allocation is the automatic allocation of memory at run-time

- It is accomplished by two functions: `malloc` and `free`

- These functions are defined in the library file `stdlib.h`

- `malloc` allocates the specified number of bytes and returns a pointer to the block of memory

- When the memory is no longer needed, the pointer is passed to `free` which deallocates the memory

- Other functions:
  - `calloc` allocates the specified number of bytes and initializes them to zero
  - `realloc` increases the size of the specified chunk of memory

*Note: With arrays, static memory allocation takes place, that is at compile-time.*
Example: dynMemAlloc.c (1)

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

int main(){
    int numStudents, avg, *ptr, i, sum = 0;
    printf("Enter the num of students : ");
    scanf("%d",&numStudents);
    ptr=(int *)malloc(numStudents*sizeof(int));
    if(ptr == NULL){
        printf("\n\nMemory allocation failed!");
        exit(1);
    }
    for (i=0; i<numStudents; i++){
        printf("\nEnter the marks for the student %d\n", i+1);
        scanf("%d",(ptr+i));
    }
```
Example: dynMemAlloc.c (2)

```c
... 
for (i=0; i<numStudents; i++){
    sum = sum + *(ptr + i);
}
avg = sum/numStudents;
printf("\nAvg marks = %d ",avg);
return 0;
} // end of main function
```

Output:
Enter the num of students : 3
Enter the marks for the student 1
  10
Enter the marks for the student 2
  20
Enter the marks for the student 3
  30
Avg marks = 20
Overview of Content

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

All the concepts are accompanied by examples.
Including Library File for Maths:
mathExample.c

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

int main()
{
    double myNum = 2.2;
    int times = 8;
    printf("Square root of %lf is: %lf\n", myNum, sqrt(myNum));
    return 0;
}

Output:
Square root of 2.200000 is: 1.483240
User-Defined Header Files

• Useful in multi-module, multi-person software development effort
• Save the following code in a file named head.h and don’t compile/run it

/* This is my little header file named head.h */
#define HAPPY 100
#define SPIT printf
#define POOL {
#define PEEL }
User-Defined Header Files

- This is how the file head.h can be included in any program, here headTest.c

```c
#include <stdio.h>
#include "head.h" /* Notice the quotes around file name
int main()
POOL
SPIT("This guy is happy: %d percent\n", HAPPY);
return(0);
PEEL
```

Output:
This guy is happy: 100 percent
File I/O

- File pointer is required for accessing files to read, write or append
  ```c
  FILE *fp;
  ```

- `fopen` function is used to open a file and it returns a file pointer
  ```c
  FILE *fopen(const char *filename, const char *mode);
  ```

- The modes in which a file can be opened
  - r  - open for reading
  - w  - open for writing (file need not exist)
  - a  - open for appending (file need not exist)
  - r+ - open for reading and writing, start at beginning
  - w+ - open for reading and writing (overwrite file)
  - a+ - open for reading and writing (append if file exists)

- To close a file
  ```c
  int fclose(FILE *a_file);
  ```
```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int i, myInt;
    FILE *ifp;
    char *mode = "r";
    ifp = fopen("in.txt", mode);
    if (ifp == NULL) {
        fprintf(stderr, "Can't open input file in.txt!\n");
        exit(1);
    } else {
        for (i=0; i<10; i++) {
            fscanf(ifp, "%d", &myInt); <-- fscanf is used for reading file contents
            printf("%d\n", myInt);
        }
    }
    fclose(ifp);
    return 0;
}
```
#include <stdio.h>

int main()
{
    FILE *fp;
    fp = fopen("in2.txt", "a+");
    fprintf(fp, "\n%d", 7000);
    fclose(fp);
    return 0;
}
References

• C Programming Language, Brian Kernighan and Dennis Ritchie

• Let Us C, Yashavant Kanetkar

• C for Dummies, Dan Gookin

• http://cplusplus.com