C Programming Basics

SDS 322/392

November 19, 2015

Email: rauta@tacc.utexas.edu

Overview of the Lecture

- 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
- Misc. Topics

All the concepts are accompanied by examples.

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

Example: dynMemAlloc.c (1)

```
#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
```

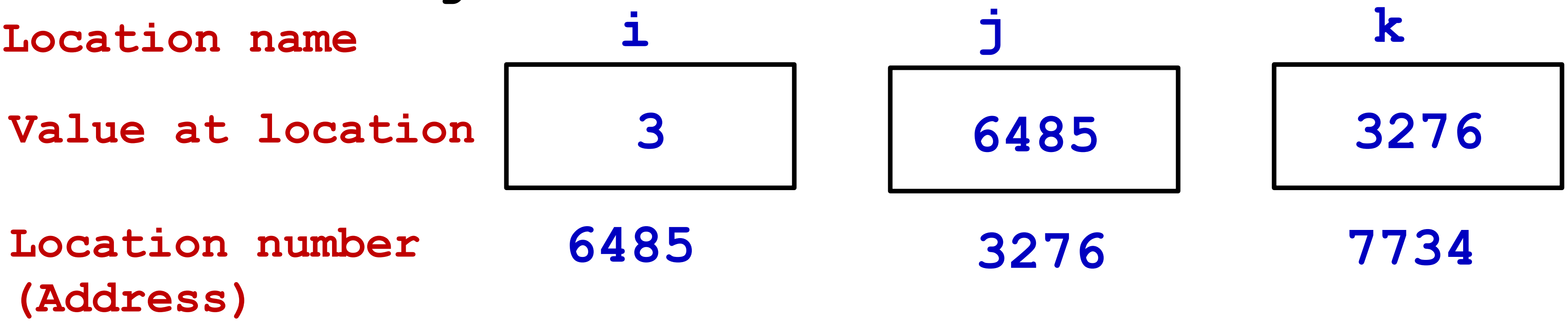
```
Output:
Enter the num of students :3
Enter the marks for the student 1
10
Enter the marks for the student 2
20
Enter the marks for the student 3
30
Avg marks = 20
```

Pointer-to-a-Pointer

- As you know, pointer is a variable that contains the address of another variable
  - this variable could as well be another pointer!
  - in this case, a pointer contains the address of another pointer

```
int i = 3;
int *j = &i;
int **k = &j;
```

//Here **k** is a pointer-to-a-pointer



Location name	i	j	k
Value at location	3	6485	3276
Location number (Address)	6485	3276	7734

Pointer-to-a-Pointer & Multi-Dimensional Arrays

- We have used the function **`malloc`** for dynamically allocating a block of memory to work like an array in one-dimension

- The concept of pointer-to-a-pointer can be used along with **`malloc`** to dynamically allocate memory for multi-dimensional arrays in C

- If you do not know how many columns a 2-D array will have at compile-time then
  - First allocate the memory for each row by calling `malloc` such that each row is represented by a pointer; you will need to create an array of pointers to keep track of pointers for each row!
  - We do not know how many rows there will be either, hence the array of pointers is implemented with a pointer-to-a-pointer

Function to allocate memory for a 2-D array of type double

```
double** allocArrayDouble(double **a, int M, int N){
  int i;
   /*  allocate storage for an array of pointers */
  a = malloc(M * sizeof(double *));

  /* for each pointer, allocate storage for an array
   of double */
  for (i = 0; i < M; i++) {
    a[i] = malloc(N * sizeof(double));
  }

  return a;
}
```

Matrix-Multiplication using Functions and Dynamic Memory Allocation (1)

```
#include <stdio.h>
#include <stdlib.h>
//note the names of the matrices being passed to fcts.
void printMatrix(double **arrayA, int M, int N);
void matmul(double **matA,double **matB, int M,int N);
double** allocArrayDouble(double **a, int M, int N);

int main(){
    int Rows, Cols;
    int i, j, count;
    double **myMatrixA, **myMatrixB;
     Rows = 2;
     Cols = 2;
```

Matrix-Multiplication (2)

```
    myMatrixA = allocArrayDouble(myMatrixA, Rows, Cols);
    myMatrixB = allocArrayDouble(myMatrixB, Rows, Cols);
    count = 1;
    for(i=0; i<Rows; i++){
          for (j=0; j<Cols; j++){
             myMatrixA[i][j] = count;
             myMatrixB[i][j] = count;
             count++;
          }
    }
    matmul(myMatrixA, myMatrixB, Rows, Cols);
    free(myMatrixA);
    free(myMatrixB);
    return 0;
}
```

Matrix Multiplication (3)

```
void matmul(double **matA, double **matB, int M, int N)
{
  int i,j,k,sum;
  double **myMatrixC =
                allocArrayDouble(myMatrixC,M,N);
   //perform matrix multiply
   for (i = 0; i < M; i++){
       for (j = 0; j < N; j++){
           sum = 0;
           for (k = 0; k < N; k++){
               sum += matA[i][k] * matB[k][j];
           }
           myMatrixC[i][j] = sum;
       }
   }
```

Matrix Multiplication (4)

```
 printMatrix(myMatrixC, M, N);
 free(myMatrixC);
}
void printMatrix(double **arrayA, int M, int N){
 int i,j;
  printf("Matrix C is:\n");
  for (i = 0; i < M; i++){
      for (j = 0; j < N; j++){
          printf(" %lf ",arrayA[i][j]);
      }
      printf("\n");
  }
  printf("\n");
}
```

Matrix Multiplication(5)

```
double** allocArrayDouble(double **a, int M, int N){
  int i;
  a = malloc(M* sizeof(double *));
  //check for NULL here
  for (i = 0; i < M; i++) {
    a[i] = malloc(N * sizeof(double));
    // check for NULL here
  }
  return a;
}
```

Confused?

- With the latest implementation of C standard, you can declare an array of length “N” at run-time such that you first read “N” as an input to the program
  - That is, postpone the declaration of array till you know its length

```
int main(){
    int Rows, Cols;
    printf("Enter the value for number of rows:\n");
    scanf("%d", &Rows);
    printf("Enter the value for number of cols:\n");
    scanf("%d", &Cols);
    int myArray[Rows][Cols];
    //other stuff goes after this
}
```

Accessing Array Elements Using Pointers

```
#include <stdio.h>
int main(){
    int a[5][5];
    int i, j;
    for(i=0; i <5; i++){
      for(j=0; j <5; j++){
        a[i][j] = i*j;
      }
    }
    printf("Value pointed to by ptr is: %d, %d\n",
a[1][1],*(*(a+1)+1) );

    return 0;
}
```

Reflect on this and let me know by the next class if you can connect the dots.

References

- Pointers in C, Yashavant Kanetkar
- http://www.eskimo.com/~scs/cclass/int/sx9b.html
- http://stackoverflow.com/questions/1169858/global-memory-management-in-c-in-stack-or-heap