BM267 - Introduction to Data Structures

2. Elementary Data Structures Part 1

Ankara University

Computer Engineering Department

Bulent Tugrul

Objectives

- Review basic C knowledge
  - Learn basic types, int, float, char.
  - Learn how to write and call functions.
  - Learn how to define C structures which put pieces of information together.
  - Learn to use pointers which refer to information indirectly.
  - Learn general approach to organize our C programs.

Basic Types

- A data type is a set of values and collection of operations on those values.
- In C, programs are built from just a few types of data
  - Integers:short int, int, long int
  - Floating-point numbers: float, double
  - Characters:char
- When we perform an operation, we need to ensure that its operands and result are of the correct type.
- C performs implicit type conversions.
- We can use cast, or explicit type conversions.
- For example, if x and N are integers

  ((float) x) / N the result of this operation is floating point

Operations on basic data types

Arithmetic operations `+ - * / % ++ --`

Relational operations `== < > != <= >=`

Logical operations `&& ||`

Bitwise operations `& | ^ ~`

Shift operations `<< >>`

Functions

- We define functions to implement new operations on data.
- All C programs include a definition of the function main().
- All functions have a list of parameters, the list can be empty and functions may return a value or nothing.
- In order to declare a function, you should give return type, its name and paramater types.
- Ex: int lg(int);
- In a function definition, you should give names to the arguments, do the desired computation using these parameters.
- Definition and declaration of function could be in different files, but you should include the declaration file into definition file.

Functions Example

```
#include <stdio.h>
int lg(int);
int main(){
   int i, N;
       for (i = 1, N = 10; i <= 6; i++, N *= 10)
             printf("%7d %2d %9d\n", N, lg(N),
   N*lg(N));
   return 0;
}
int lg(int N){
   int i;
      for (i = 0; N > 0; i++, N /= 2) ;
      return i;
}
```

Functions Example –2 Average and Standard Deviation of N integers

```
1.   #include <stdlib.h>
2.   #include <stdio.h>
3.   #include <math.h>
4.   typedef int numType;
5.   numType randNum()
6.     { return rand(); }
7.   int main(int argc, char *argv[])
8.     { int i, N = atoi(argv[1]);
9.       float m1 = 0.0, m2 = 0.0;
10.      numType x;
11.      for (i = 0; i < N; i++)
12.        {
13.          x = randNum();
14.          m1 += ((float) x)/N;
15.          m2 += ((float) x*x)/N;
16.        }
17.      printf("       Average: %f\n", m1);
18.      printf("Std. deviation: %f\n", sqrt(m2-m1*m1));
19.   }
```

Program Organization

- As it is recommended, you can split your program into three files.
- **.h file**: An interface, which defines the data structure and declares the funtions to be used to manipulate.
- **.c**: An implementation of the functions declared in the .h file (must include .h file).
- A client program that uses the functions declared in the interface (must include .h file). This file must implement main() function.

Program Organization (2)

Num.h

```
1.   typedef int numType;
2.   numType randNum();
```

Num.c

```
1.   #include <stdlib.h>
2.   #include "Num.h"
3.   numType randNum()
4.     { return rand(); }
```

Client.c

```
1.   #include <stdio.h>
2.   #include <math.h>
3.   #include "Num.h"
4.   int main(int argc, char *argv[])
5.   {  implementation of main }
```

Structs

- We need data structures that allow us to handle collections of data. Arrays and struct allow us to organize data.
- Structs define a new type of data.
- Structs are aggregate types that we use to define collections of data. The members of a struct can be different type, it can even be another struct, but arrays can hold only one type of data.
- Assume that we need a new type which is callled Point, unfortunately, there is no such a built-in type in C standard.
- But C allows us to define such a mechanism using "***struct***".

Structs(2)

- Accordingly, we can write;

  `struct Point {float x; float y;};` ←Do not forget the semicolon.
- `struct Point a, b;` declares two Point variables.
- We can refer each member of the Point struct by their names. For example

  ```
  a.x=1.0; a.y=1.0; b.x=4.0;
  b.y=5.0;
  ```
- We can also pass structs as arguments of a functions. For example

Structs(3)

Point.h

- `typedef struct { float x; float y; } point;`
- `float distance(point a, point b);`

Point.c

- `#include <math.h>`
- `#include "Point.h"`
- `float distance(point a, point b)`
- `  { float dx = a.x - b.x, dy = a.y - b.y;`
- `    return sqrt(dx*dx + dy*dy);`
- `  }`

Pointers

- C pointers provides us to manipulate data indirectlly. Basically pointer is a reference to an object in the memory.
- In order to declare a pointer, you should first give its type and then put a “*” before giving the variables name. Ex `int *a_Ptr;`
- We can declare pointers to any type of data.

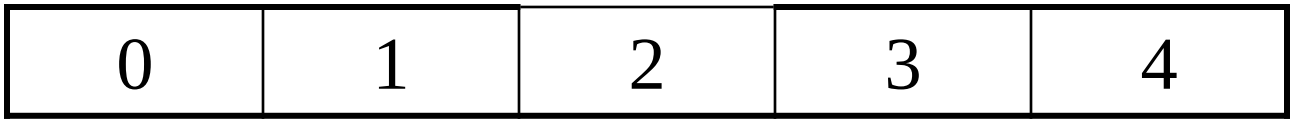
  Ex: `float *f_Ptr, Point *point_Ptr.`
- The “&” operator returns the adreess of a variable.
- When you want to initialize a pointer you can use “&” operator. Ex `int a, *a_Ptr=&a;`

Pointers(2)

- C functions returns only one value, but pointers allow us to manipulate more variables.
- Ex:
```
polar( float x, float y, float*r, float* theta)
{
    *r = sqrt(x*x+y*y);
    *theta= atan2(y,x);
}
```
- The function call polar(1.0, 1.0, &a, &b) will effect the values of a and b. ‘a’ will become sqrt(x*x+y*y) and ‘b’ will become atan2(y,x);

Arrays

- An array is fixed collection of same type data that are stored contiguously in the memory.
- a



- You can declare an array in this way: type name[const unsigned int];
- You can reach the element of an array by its index. Ex: a[i];

Dynamic Memory Allocation

- Dynamic memory allocation allow us to obtain blocks of memory as needed during execution.
- Using dynamic memory allocation, we can design data structures that grow and shrink.
- To allocate memory dynamically, we will need to call one of the three memory allocation functions declared in the <stdlib.h> header.
  - malloc: allocates a block of memory, but doesn't initialize it.
  - calloc: allocates a block of memory and clears it.
  - realloc: Resizes a previously allocated block of memory.
- malloc returns a value of type void*
- When we call a memory function, it may not allocate enough memory, in this case it returns NULL, we must test this situation.

Dynamic Memory Allocation(2)

- Ex: `p = malloc( 10000 );`

  `İf( p==NULL)`

  `// allocation failed, take appropriate action;`
- We use ***sizeof*** operator to calculate the amount of space required.
- Ex: `Point * p = malloc(sizeof(Point))` or

  `Point * p = malloc( n* sizeof(Point))` it allocates n Point object.
- Once it points to a dynamically allocated block of memory, we can use p as if it is an array.
- ```
  for( int i = 0; i < n; i++)
     p[i].x = i;
     p[i].y = i;
  ```
  ```

Dynamic Memory Allocation(3)

- **`free()`** Deallocates memory allocated by **`malloc`**
- Takes a pointer as an argument
- `free ( ptr );`