DSA
INTERNSHALA
Revision of relevant topics in C

Pointers

A pointer variable is a special kind of variable that stores memory address of a variable or an array. A pointer variable is not used to store ordinary data.

There are many uses of pointers:

- They come in handy when programmers are required to apply dynamic memory allocation.
- The pointers come in handy while passing arguments to a function using call by reference.
- If a pointer variable contains address of an int variable or array element then it's called 'pointer to int'
- If a pointer variable contains address of a float variable or array element then it's called 'pointer to float'
- Similarly we can have 'pointer to long int', 'pointer to double' and 'pointer to char' and so on.

Syntax of declaring a pointer variable:

```
<data type> *<pointer name>;     //declaration of pointer
```

```
int *pt;           //declaring pointer to int
int n = 100;
pt = &n;           //assigning address of n into pointer variable pt
```

Pointer operators:

```
&     address of operator
*     value of address operator / dereference operator / indirection operator
**    pointer to pointer operator
%u    to print address of a variable which is unsigned int
```

Pointer arithmetic:

Operations like `++pt` or `pt++` does not add 1 to the value stored in the address stored in `pt` instead it adds the size of the data type of the pointer to its value (address stored in it) thus for a `pt` of type float on doing `++pt` the pointer `pt` will refer to the (previous address + 4 bytes).

Operation like `&arr[5] - &arr[3]` will not be difference between their address but result would be the difference divided by the size of the data type so for example it would be 2.

Arrays and pointers:

When we declare an array then a pointer variable of same name as that of array name is declared and is assigned the base address.

So, we can say that an array name is a pointer to the base address.

So when we do arr[n] we are actually adding n to base address of array which according to pointer arithmetic will mean that `arr + (n*data type size)`.

Dynamic memory allocation:

Allocating space to an array dynamically:

```
int *arr, n;
arr = (int *) malloc(sizeof(int)*n); // n is inputted from user before this line

arr = (int *) realloc(arr, sizeof(int)*new_n);    // realloc can be used to reallocate size of array
```

Functions

- A function is a sub program which performs a logically isolated task.
- The whole program is broken down into various functions thus making the readability and maintainability of the program highly improved.

- We can draw an analogy with an article or write up being divided into various paragraphs so that user finds it convenient to read and understand.
- When we call a function the control is transferred to it's body where all it's statements are executed and then control returns back.
- The body of a function comprises one or more programming statements. The alternative name for function body is function definition.
- Functions are categorized as:
    - Library functions (in-built functions) (contained in header files)
    - User Defined functions (written by programmer)
- Components of a function
    - Function prototype / Function declaration
        - Used to denote the data type of the value it will return, function name, data types of passing arguments.
    - Function Calling / Function Invoking
        - This statement calls the function by specifying it's name and passing arguments, if any.
    - Function Definition / Function Body
        - It consists of all the statements that constitute the function.

Scope of variables:

- Local variables
- Global variables

Local variables get precedence over global variables.

In case there is a clashing between local and global variables with same name by default local variable gets precedence. But we can still access the global variable by using the `extern` keyword.

```
int n = 10;
int main() {
int n = 90;
 {
      extern int n;
      printf("%d",n);
 }
}
```

Output for the above code will be 10.

return keyword returns only one value so if multiple values are passed to be returned than only the last value will be returned.

```
return z , y , x;  // this will return x
```

C does not support function overloading so functions with same name but different parameter list will generate an error.

A function can be called in two ways:

- Call by value
  - A copy of original variables is passed and any change made in them does not reflects on actual arguments.
  - Just pass the variable name while calling and declare a normal variable in function prototype
- Call by reference
  - A reference to original variables is passed and all the changes made reflect on actual arguments.
  - Pass the address of variables using & and declare pointer variables using * in function prototype

Passing arrays in function:

Array name is by default a pointer to base address so when we pass just the name of array the base address is passed so the function prototype should be a pointer.

Structures

A structure is a finite set of heterogeneous or homogeneous elements stored in contiguous memory locations.

A structure is a user defined data type.

There can be multiple structure variables for one structure specification.

Components of a structure:

- Structure specification
- Declaration of structure variable

```
struct customers {
```

```
   int custId;
char name[25];
unsigned long cellno;
float balance;
};                          // structure specification

struct customers cvar;      // structure declaration
```

Member access operators:

- Dot operator .
  - To access members of structures
- Arrow operator ->
  - To access members of structures being referred to as pointers

We can declare an array of structures if we need to declare multiple structures to avoid creating multiple declarations with separate names.

```
struct customers {
   int custId;
char name[25];
unsigned long cellno;
float balance;
} cvar;  // structure can also be declared like this
```

```
struct customers {
   int custId;
char name[25];
unsigned long cellno;
float balance;
} cvar, cvar2, cvar 3;      // structure can also be declared like this
```

```
struct customers {
   int custId;
char name[25];
unsigned long cellno;
float balance;
} cvar[10];    // structure can also be declared like this
```

A structure variable can be passed to functions just like how we pass normal variables by call by value or reference.