INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

CSN-101 (Introduction to Computer Science and Engineering)

Lecture 19: Basics of C Programming

Ms. Sumit Sharma and Mr. Debraj Kundu

Dr. Sudip Roy

Assistant Professor, Department of Computer Science and Engineering

Piazza Class Room: https://piazza.com/iitr.ac.in/fall2019/csn101

[Access Code: csn101@2019]

Moodle Submission Site: https://moodle.iitr.ac.in/course/view.php?id=45

[Enrollment Key: csn101@2019]

Functions

- Self-contained block of statements that can be executed repeatedly
- A fragment of code that accepts zero or more *argument values*, produces a *result value*
- A method of *encapsulating* a subset of a program or a system
  - To hide details
  - To be invoked from multiple places
  - To share with others

Functions in C

resultType **`functionName(`***type*$_1$ **`param1,`** *type*$_2$ **`param2, …)`**

```
{
    …
    body
    …
}
```

- If no result, *resultType* should be **`void`**
  - Warning if not!
- If no parameters, use **`()`**

Functions in C

- *Parameter:–* a declaration of an identifier within the `'()'` of a function declaration
  - Used within the body of the function as a *variable* of that function
  - Initialized to the value of the corresponding *argument*.
- *Argument:–* an expression passed when a function is *called*; becomes the initial value of the corresponding parameter

This is a *Parameter*

Let `int f(double x, int a)` be (the beginning of) a declaration of a function.

Then we can call this function as:

$f(expr_1, expr_2)$

This is an *Argument*

Function Types

- Build-in (Library) Functions:
  - The system provided these functions and stored in the library. Therefore it is also called *Library Functions*.
    e.g. scanf(), printf(), strcpy, strcmp, strlen, sin(x), cos(x), etc.
  - To use these functions, you just need to include the appropriate C header files.
- User Defined Functions:
  - These functions are defined by the user at the time of writing the program.

Function Calling

- Addition of two number using user defined function

```
/* function returning the addition of two numbers */
int addition()
{
    int a = 2, b =3;
    return a + b;
}

int main()
{
     int answer;
     /* calling a function to get addition value */
     answer = addition();
     return 0;
}
```

Recursion

- In some problems, it may be natural to define the problem in terms of the problem itself
- Recursion is useful for problems that can be represented by a simpler version of the same problem.
- Recursion *splits a problem*:
  - Into one or more *simpler versions of* ***itself***
- Example: the factorial function

```
6! = 6 * 5 * 4 * 3 * 2 * 1
```

We could write:

```
6! = 6 * 5!
```

General Approach for Recursion

1. if problem is "*small enough*"
2. solve it *directly*
3. else
4. break into one or more *smaller subproblems*
5. solve each subproblem *recursively*
6. *combine* results into solution to whole problem

Recursion Example

In general, we can express the factorial function as follows:

```
n! = n * (n-1)!
```

The factorial function is only defined for *positive* integers.

So we should be a bit more precise:

```
n! = 1              (if n is equal to 1)
n! = n * (n-1)!     (if n is larger than 1)
```

C equivalent definition:

```
int fac(int n)
{
if(n<=1)
return 1;

else
return n*fac(n-1);
}
```

recursion **means that a function calls itself**

Recursion

If we use recursion we must be careful not to create an infinite chain of function calls:

```
int fac(int numb)
{
   return numb * fac(numb-1);
}
```

No termination condition

Or:

```
int fac(int numb)
{
   if (numb<=1)
      return 1;
   else
      return numb * fac(numb+1);
}
```

Incorrect termination condition

Pointers

- A pointer is a variable that represents the location (rather than the value) of a data item

- They have a number of useful applications.
  - Enables us to access a variable that is defined outside the function.
  - Can be used to pass information back and forth between a function and its reference point.
  - Used in dynamic (on-the-fly) memory allocation (especially of arrays)
  - Used in building complex data structures (linked lists, stacks, queues, trees, etc.)

Pointers Variable Definition

Basic syntax:

*Type *Name*

Examples:

```
int *P;          /* P is var that can point to an int var */
float *Q;        /* Q is a float pointer */
char *R;         /* R is a char pointer */
```

Complex example:

```
int *A[5];       /* A is an array of 5 pointers to ints */
```

Address Operator

The address (&) operator can be used in front of any variable object in C -- the result of the operation is the location in memory of the variable

Syntax: *&VariableReference*

Examples:

```
int V;
int *P;
int A[5];
```

&V - memory location of integer variable V

&(A[2]) - memory location of array element 2 in array A

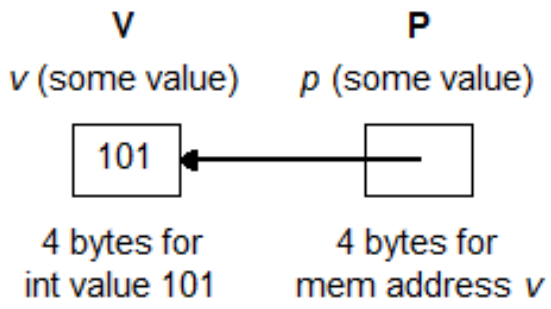
&P - memory location of pointer variable P

Pointer Basics

- Variables are allocated at *addresses* in computer memory
- Name of the variable is a reference to that memory address
- A pointer variable contains a representation of an address of another variable (P is a pointer variable in the following):

```
int V = 101;

int *P = &V;
```



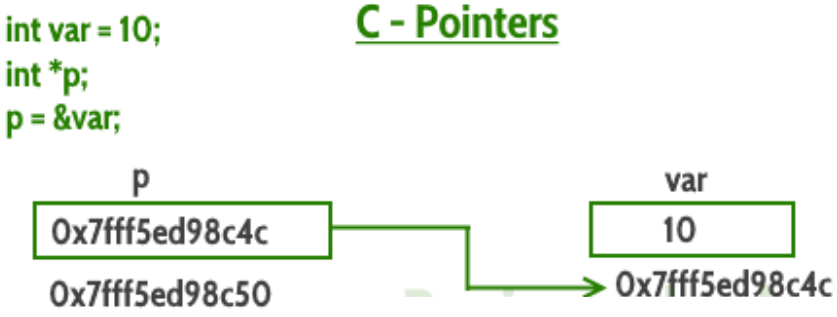
Pointer Example

```c
#include <stdio.h>
int main()
{
   /* Pointer of integer type, this can hold the
    * address of a integer type variable.
    */
   int *p;

   int var = 10;

   /* Assigning the address of variable var to the pointer
    * p. The p can hold the address of var because var is
    * an integer type variable.
    */
   p= &var;

   printf("Value of variable var is: %d", var);
   printf("\nValue of variable var is: %d", *p);
   printf("\nAddress of variable var is: %p", &var);
   printf("\nAddress of variable var is: %p", p);
   printf("\nAddress of pointer p is: %p", &p);
   return 0;
}
```



Output

```
Value of variable var is: 10
Value of variable var is: 10
Address of variable var is: 0x7fff5ed98c4c
Address of variable var is: 0x7fff5ed98c4c
Address of pointer p is: 0x7fff5ed98c50
```

Pointer Arithmetic

- A pointer in C is an address, which is a numeric value.
- Following arithmetic operations can be performed on a pointer : ++, --, +, and -

```
#include<stdio.h>
int main(){
int number=50;
int *p;//pointer to int
p=&number;//stores the address of number variable
printf("Address of p variable is %u \n",p);
p=p+1;
printf("After increment: Address of p variable is %u \n",p);
return 0;
}
```

Output

```
Address of p variable is 3214864300
After increment: Address of p variable is 3214864304
```

Pointer Scale Factor

- An Integer value can be added and subtracted from a pointer variable.
- In reality, it is not the integer value which is added/subtracted, but rather the scale factor times the value.

Data Type	Scale Factor
char	1
int	4
float	4
double	8

– If p1 is an integer pointer, then

p1++

will increment the value of p1 by 4.

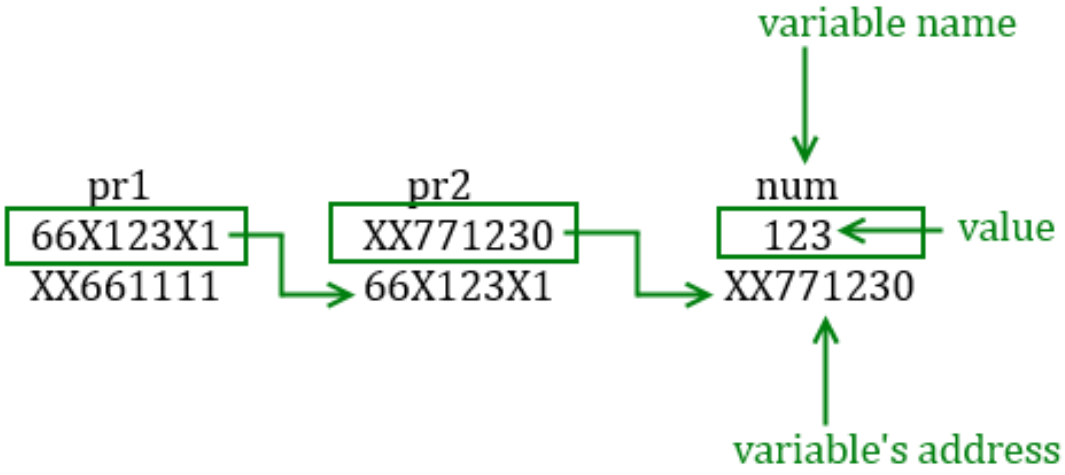
Pointer to Pointer

A pointer can also be made to point to a pointer variable (but the pointer must be of a type that allows it to point to a pointer)

Syntax:

- int **pr; /* Here pr is a double pointer */

- Example:




```
Variable num has address: XX771230
Address of Pointer pr1 is: XX661111
Address of Pointer pr2 is: 66X123X1
```

Pointer and Arrays

- When an array is declared,
  - The compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in contiguous memory locations.
  - The base address is the location of the first element (index 0) of the array.
  - The compiler also defines the array name as a constant pointer to the first element.

Pointer and Arrays: Example

Consider the declaration:

int x[5] = {1, 2, 3, 4, 5} ;

– Suppose that the base address of x is 2500, and each integer requires 4 bytes.

Element	Value	Address
x[0]	1	2500
x[1]	2	2504
x[2]	3	2508
x[3]	4	2512
x[4]	5	2516

x ⇔ &x[0] ⇔ 2500 ;

p = x; and p = &x[0]; are equivalent.

Relationship between p and x:

p = &x[0] = 2500
p+1 = &x[1] = 2504
p+2 = &x[2] = 2508
p+3 = &x[3] = 2512
p+4 = &x[4] = 2516

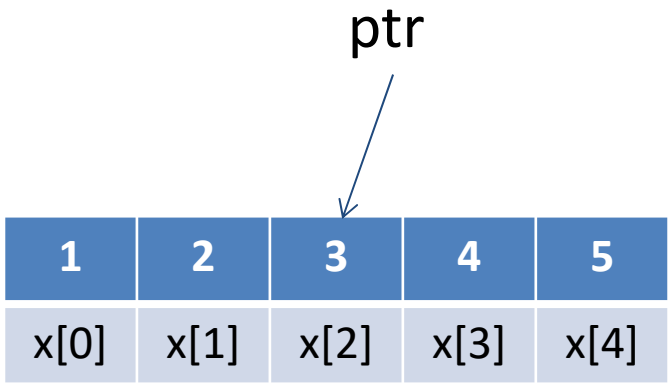
Pointer and Arrays

```c
#include <stdio.h>
int main()
{
   int x[5] = {1, 2, 3, 4, 5};
   int* ptr;

   // ptr is assigned the address of the third element
   ptr = &x[2];

   printf("*ptr = %d \n", *ptr);    // 3
   printf("*ptr+1 = %d \n", *ptr+1); // 4
   printf("*ptr-1 = %d", *ptr-1);   // 2

   return 0;
}
```



Passing Pointers to Function

- Declare the function parameter as a pointer type

```c
#include <stdio.h>

void addOne(int* ptr) {
   (*ptr)++; // adding 1 to *ptr
}

int main()
{
   int* p, i = 10;
   p = &i;
   addOne(p);

   printf("%d", *p); // 11
   return 0;
}
```

Call by Value and Call by Reference

Call By Value

```
#include<stdio.h>
void change(int num) {
    printf("Before adding value inside function num=%d \n",num);
    num=num+100;
    printf("After adding value inside function num=%d \n", num);
}
int main() {
    int x=100;
    printf("Before function call x=%d \n", x);
    change(x);//passing value in function
    printf("After function call x=%d \n", x);
return 0;
}
```

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=100
```

Call by Value and Call by Reference

Call By Reference

```
#include<stdio.h>
void change(int *num) {
    printf("Before adding value inside function num=%d \n",*num);
    (*num) += 100;
    printf("After adding value inside function num=%d \n", *num);
}
int main() {
    int x=100;
    printf("Before function call x=%d \n", x);
    change(&x);//passing reference in function
    printf("After function call x=%d \n", x);
return 0;
}
```

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=200
```

Introduction to Algorithm

- Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to solve some problem

- Example: To find the max and min item from the set of *n* elements

```
MaxMin(A, n, max, min)
// Set max to the maximum and min to the minimum of A[1..n]
{
        max = min = A[1];
        for(i = 2 to n) do
        {
                if (A[i] > max) then max = A[i];
                 if (A[i] < min) then min = A[i];
        }
}
```

Thanks!