

Pointers in C

Lecture 09 – ICT1132



Piyumi Wijerathna
Department of ICT
Faculty of Technology

Overview

- What are pointers?
- Pointers and function arguments
- Command Line Arguments in C
- Arrays and Pointers
- Dynamic Memory allocation

Introduction

- Every variable is a memory location.
- Every memory location has its address defined which can be accessed using ampersand (&) operator. (which denotes an address in memory)
- If you have a variable *var* in your program, *&var* will give you its address in the memory, where & is commonly called the “reference operator”.

Example

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

```
int main ( ) {
```

```
    int var1;
```

```
    char var2[10];
```

```
    printf("Address of var1 variable: %x\n", &var1 );
```

```
    printf("Address of var2 variable: %x\n", &var2 );
```

```
    return 0;
```

```
}
```

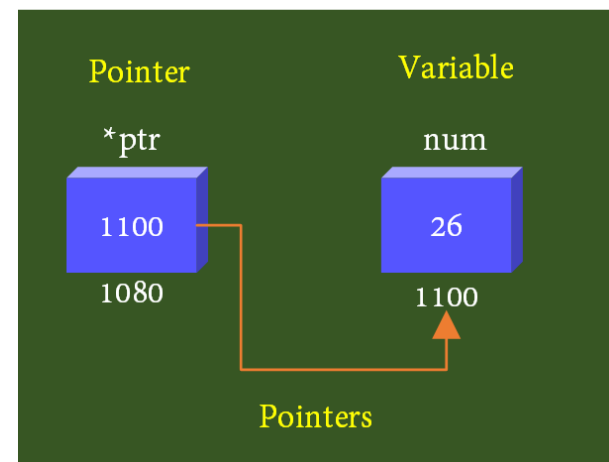
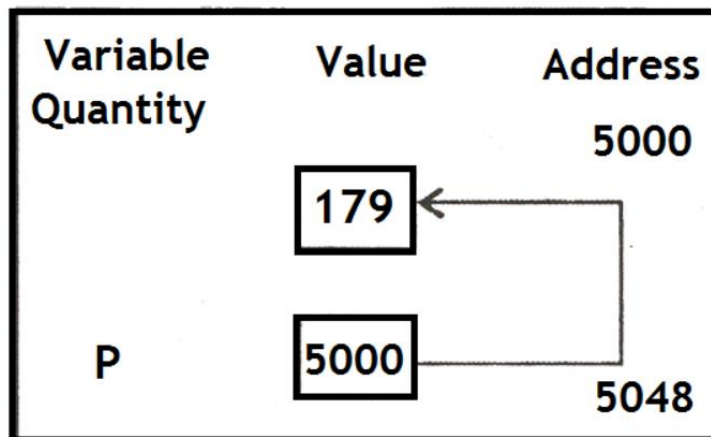
Output

Address of var1 variable: bff5a400

Address of var2 variable: bff5a3f6

What is a pointer ?

- A *pointer* is a variable that **holds a memory address**.
- Why we need pointers ?
 - Pointers reduce the length and complexity of a program.
 - They increase execution speed.
 - A pointer enables the access and modification of a variable that is defined outside the function.
 - Pointers support dynamic allocation of memory.
 - Implementing complex data structures.



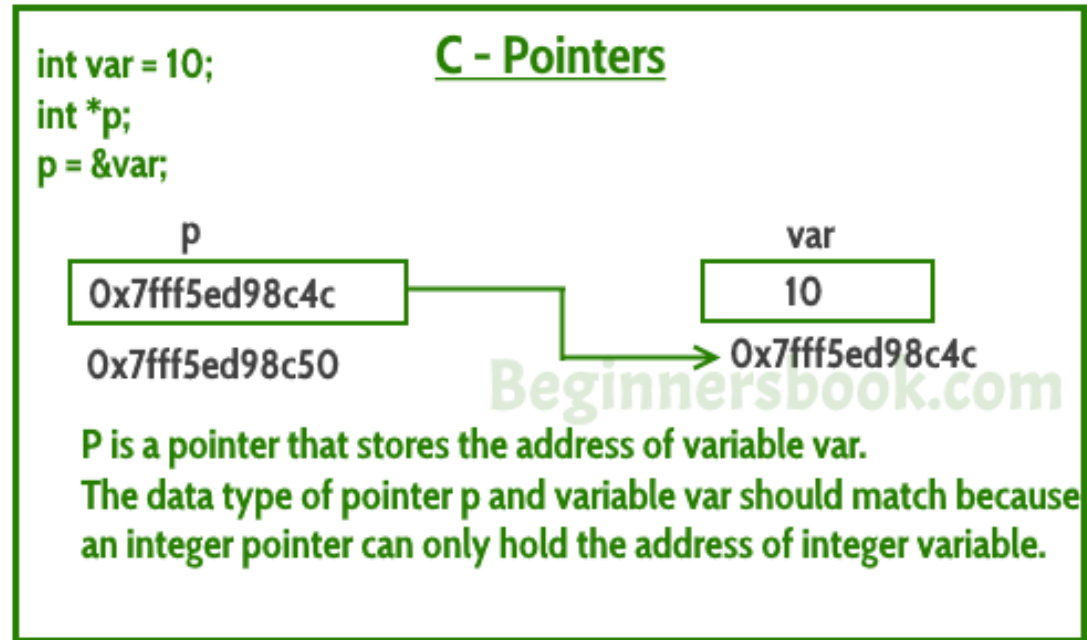
Declaring a Pointer

Syntax:

`datatype *name;`

Examples:

```
int    *ip;    /* pointer to an integer */  
double *dp;    /* pointer to a double */  
float  *fp;    /* pointer to a float */  
char   *ch     /* pointer to a character */
```

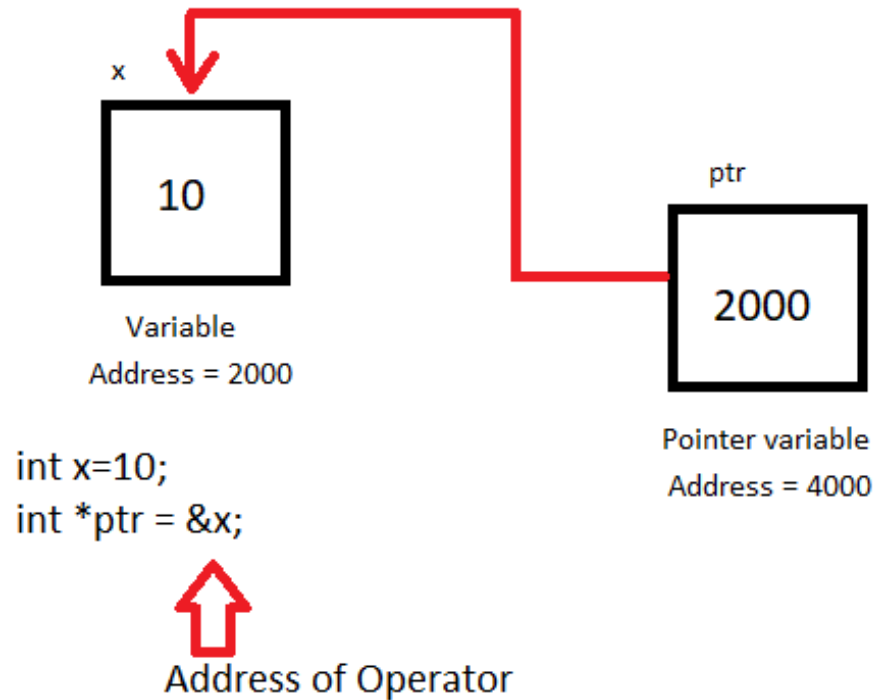


Pointer Operators

- Once a pointer is declared, the operator ***** can be used to obtain the **value located at the address** that is held by the pointer.
 - Ex-: `printf (“value of Pointer %d = “, *ip) ;`
`// It will print the value at the location pointed by ip.`
- The operator **&** can be used to obtain the memory address of an operand.
 - Ex-: `printf (“memory address %d = “, &ip) ;`
`// It will print the memory address of ip.`
- Both the operators, ***** and **&** are unary operators. That is, it uses only one operand.

& is Address operator.

* is Contents operator.




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

```
int main( ){\n
```

```
    int i = 7;\n
```

```
    int *ptr = &i;\n
```

```
    printf("content of i: %d\\n", i);\n
```

```
    printf("adrs of i: %p\\n", &i);\n
```

```
    printf("content of ptr: %p\\n", ptr);\n
```

```
    printf("value of the variable pointed by ptr: %d\\n", *ptr);\n
```

```
    printf("address of ptr: %p\\n", &ptr);\n
```

```
    return 0;\n
```

```
}\n
```

```
content of i: 7\nadrs of i: 0000000000023FE4C\ncontent of ptr: 0000000000023FE4C\nvalue of the variable pointed by ptr: 7\naddress of ptr: 0000000000023FE40
```

Pointer Operators


Declaring a pointer

- Example:

```
int *m;
```

Note:

Initially the pointer will point at (will store the memory address of) any location in the memory.



The diagram shows a curved arrow originating from the variable 'm' in the code 'int *m;' and pointing to the first row of the memory table, which contains the address 0x3267A1B0 and the content 0x3267A1B8.

Memory	
Address	Contents
0x3267A1B0	0x3267A1B8
0x3267A1B1	0x3267A1B9
0x3267A1B2	0x3267A1BA
0x3267A1B3	0x3267A1BB
0x3267A1B4	
0x3267A1B5	
0x3267A1B6	
0x3267A1B7	
0x3267A1B8	
0x3267A1B9	
0x3267A1BA	
0x3267A1BB	

Pointer Operators

- Example:

```
int *m;
```

```
int count = 7;
```

// count is a variable that stores an integer.

m

count

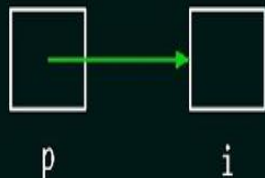
Memory

Address

Contents

0x3267A1B0	0x3267A1B8
0x3267A1B1	0x3267A1B9
0x3267A1B2	0x3267A1BA
0x3267A1B3	0x3267A1BB
0x3267A1B4	7
0x3267A1B5	
0x3267A1B6	
0x3267A1B7	
0x3267A1B8	
0x3267A1B9	
0x3267A1BA	
0x3267A1BB	
0x3267A1BC	

Pointer is a special variable that is capable of storing some address.

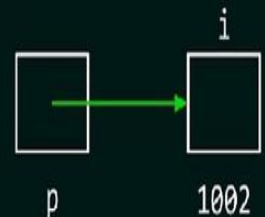


Address	contents
1000	0010 0011
1001	0101 0011
1002	0001 0010
1003	0010 1000
	⋮
	⋮
	⋮

} i

It points to a memory location where the first byte is stored

p



Pointer Operators

- Example:

```
int *m;
```

```
int count = 7;
```

```
m = &count;
```

//Assigns the memory address of
count to the pointer **m**.

- The value of **m** is 0x3267A1B4
- The value of ***m** is 7
- The value of **&m** is 0x3267A1B0

Memory	
Address	Contents
m 0x3267A1B0	0x3267A1B4
0x3267A1B1	0x3267A1B5
0x3267A1B2	0x3267A1B6
0x3267A1B3	0x3267A1B7
count 0x3267A1B4	7
0x3267A1B5	
0x3267A1B6	
0x3267A1B7	
0x3267A1B8	
0x3267A1B9	
0x3267A1BA	
0x3267A1BB	
0x3267A1BC	

Pass by Reference (Function Argument)

Pointers and Functions Arguments

- The **call by reference** method of passing arguments to a function, **copies the address of an argument** into the formal parameter.
- A pointer is passed as an argument to a function, so the address of the memory location is passed instead of the value.
- Inside the function, the address is used to access the actual argument used in the call.
- It means the changes made to the parameter affect the passed argument, not to a copy of it.

Pointers and Functions Arguments

```
void main(){  
    int x=4, y=7;  
  
    printf("x=%d and y=%d\n",x,y); //4 and 7  
  
    mySwap(&x, &y); // &x,&y address of the variables  
  
    printf("x=%d and y=%d\n",x,y); //7 and 4  
  
}
```

```
void mySwap(int *a, int *b)  
{  
    int temp;  
  
    temp = *a; // Save the value of address a  
  
    *a = *b; // put value of b into value of a  
  
    *b = temp; // put temp into b  
  
}
```

Command Line Arguments in C

Command Line Arguments in C

- It is possible to pass some values from the command line to your C programs when they are executing.
- These values are called **command line arguments**.
- User can control the program from outside instead of hard coding those values inside the code.
- The full declaration of main looks like this:

```
int main ( int argc,  char *argv[] )  
{  
  
}
```

Cont.

- Command line arguments are nothing but simply arguments that are specified after the name of the program in the system's command line.
- These argument values are passed on to your program during program execution.
- In order to implement command line arguments, generally, two parameters are passed into the main function:
 - Number of command line arguments
 - The list of command line arguments

Cont.

```
int main ( int argc,  char *argv[] )
```

int argc : Argument count.

- Number of arguments passed into the program from the command line, including the name of the program.

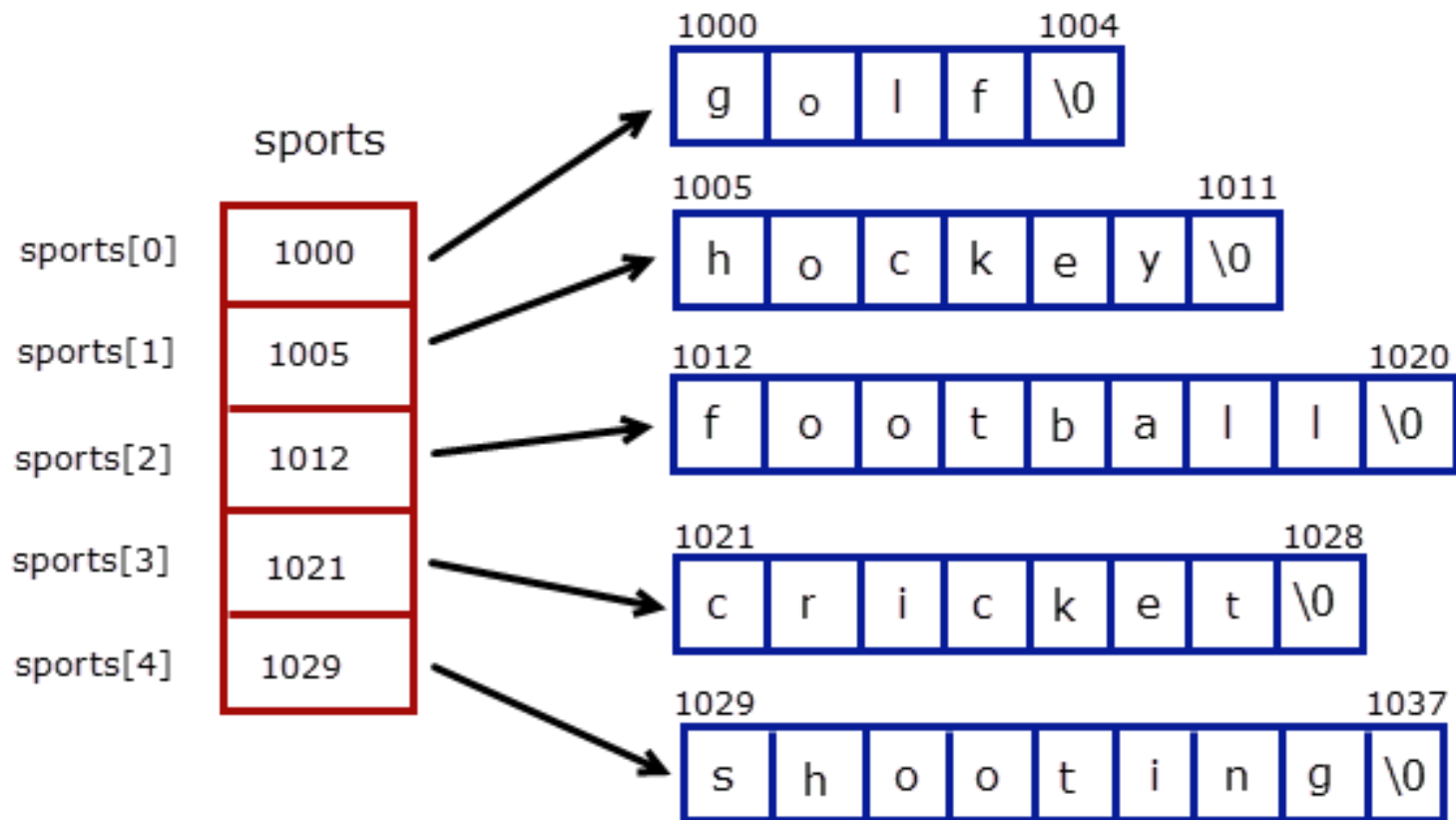
char *argv[]: Array of string pointers.

- All command-line arguments are passed to **main()** as strings.

argv[0] points to the program's name.

argv[1] points to the first argument.

Example - Array of string pointers



Example

```
#include<stdio.h>

int main(int argc, char** argv)
{
    int i;

    printf("The number of arguments are: %d\n", argc);
    printf("The arguments are:");
    for ( i = 0; i < argc; i++)
    {
        printf("%s\n", argv[i]);
    }

    return 0;
}
```

Example

```
#include <stdio.h>

int main( int argc, char *argv[] ) {

    if( argc == 2 ) {
        printf("The argument supplied is %s\n", argv[1]);
    }
    else if( argc > 2 ) {
        printf("Too many arguments supplied.\n");
    }
    else {
        printf("One argument expected.\n");
    }
}
```

When the above code is compiled and executed with single argument, it produces the following result.

```
./a.out testing
The argument supplied is testing
```

When the above code is compiled and executed with a two arguments, it produces the following result.

```
./a.out testing1 testing2
Too many arguments supplied.
```

When the above code is compiled and executed without passing any argument, it produces the following result.

```
./a.out
One argument expected
```

When the above code is compiled and executed with a single argument separated by space but inside double quotes, it produces the following result.

```
./a.out "testing1 testing2"
```

```
Program name ./a.out
```

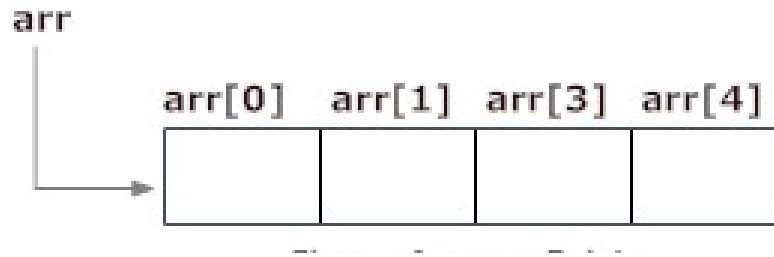
```
The argument supplied is testing1 testing2
```

Arrays and Pointers

Arrays and Pointers

- Arrays are closely related to pointers in C programming.
- An array name is a constant pointer to the first element of the array.

```
int arr[4];
```



arr and **&arr[0]** points to the address of the first element.

- **&arr[0]** is equivalent to **arr**.
- The addresses of both are the same, the values of **arr** and **&arr[0]** are also the same.
 - ✓ the address of
&arr[1] is equivalent to **(arr + 1)**
 - ✓ the value of
arr[1] is equivalent to ***(arr + 1)**

Arrays and Pointers

- An array name is a pointer.

Example:

```
//here the variable "name" is a pointer.
```

```
char name[20] = "hello";
```

- In C, to read a character we used

```
scanf ("%c", &ch) ;
```

- To read a string we need to use

```
scanf ("%s", name) ; //no & since "name is a  
pointer.
```

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

```
void main(){
```

```
    char arr[]="Hello";
```

```
    printf("%p\n",arr);
```

```
    printf("%p\n",&arr[0]);
```

```
    printf("%p\n",arr+1);
```

```
    printf("%p\n",&arr[1]);
```

```
    printf("%c\n",*arr);
```

```
    printf("%c\n",arr[0]);
```

```
    int arr1[]={10,2,3};
```

```
    printf("%p\n",arr1);
```

```
    printf("%p\n",&arr1[0]);
```

```
    printf("%p\n",arr1+1);
```

```
    printf("%p\n",&arr1[1]);
```

```
    printf("%d\n",*arr1);
```

```
    printf("%d\n",arr1[0]);
```

```
}
```

Example

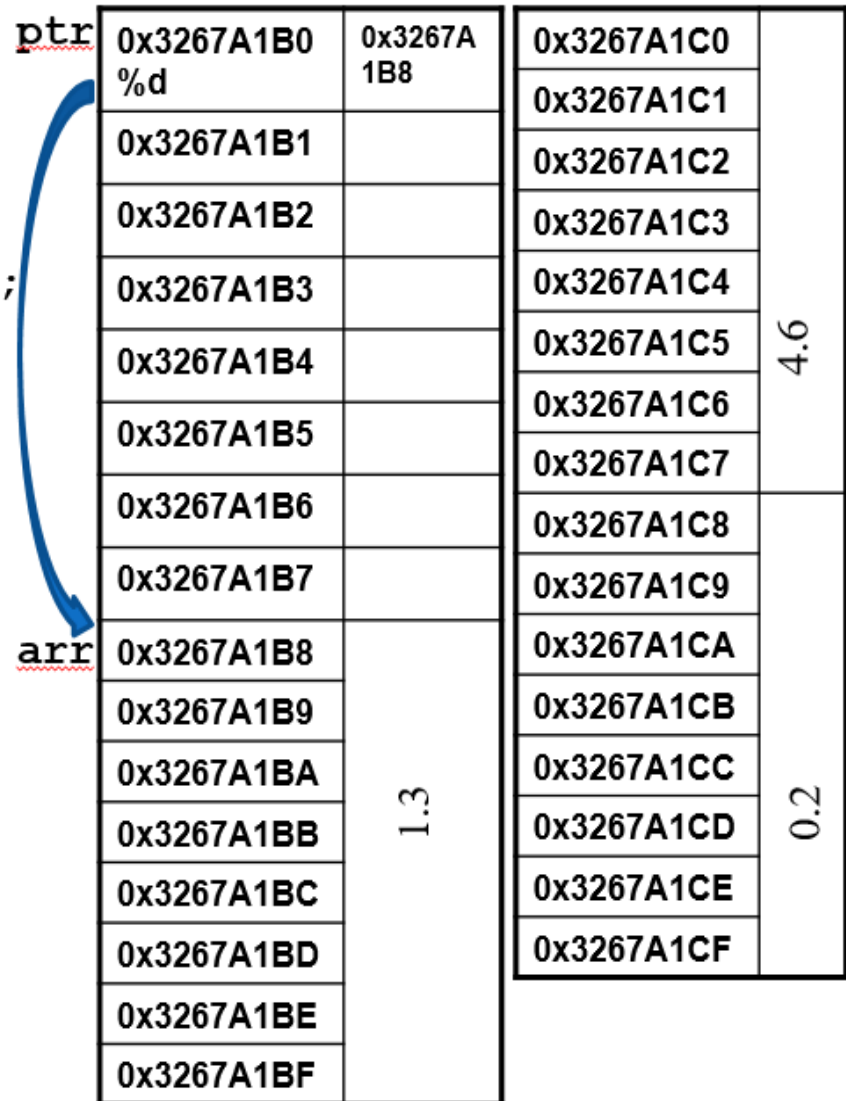
```
000000000062FE40
000000000062FE40
000000000062FE41
000000000062FE41
H
H
000000000062FE30
000000000062FE30
000000000062FE34
000000000062FE34
10
10
```

Arrays and Pointer Arithmetic

Example

```
double arr[3] = {1.3, 4.6, 0.2};  
double *ptr;  
ptr = arr;  
printf("%d", *ptr); //prints 1.3  
ptr++;  
printf("%d", *ptr); //prints 4.6  
ptr++;  
printf("%d", *ptr); //prints 0.2
```

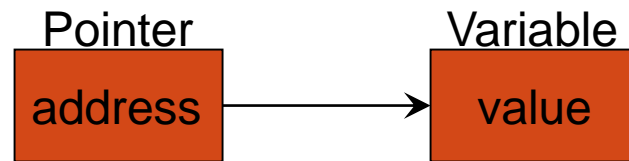
ptr	0x3267A1B0	0x3267A1B8	0x3267A1C0	4.6
	%d		0x3267A1C1	
	0x3267A1B1		0x3267A1C2	
	0x3267A1B2		0x3267A1C3	
	0x3267A1B3		0x3267A1C4	
	0x3267A1B4		0x3267A1C5	
	0x3267A1B5		0x3267A1C6	
arr	0x3267A1B6		0x3267A1C7	0.2
	0x3267A1B7		0x3267A1C8	
	0x3267A1B8	1.3	0x3267A1C9	
	0x3267A1B9		0x3267A1CA	
	0x3267A1BA		0x3267A1CB	
	0x3267A1BB		0x3267A1CC	
	0x3267A1BC		0x3267A1CD	
	0x3267A1BD		0x3267A1CE	
	0x3267A1BE		0x3267A1CF	
	0x3267A1BF			



arr++ is wrong because the base address of an array can not be changed

Multiple Indirection

- Multiple Indirection allows a pointer to point to another pointer that points to a target value.
- It also called pointers to pointers.



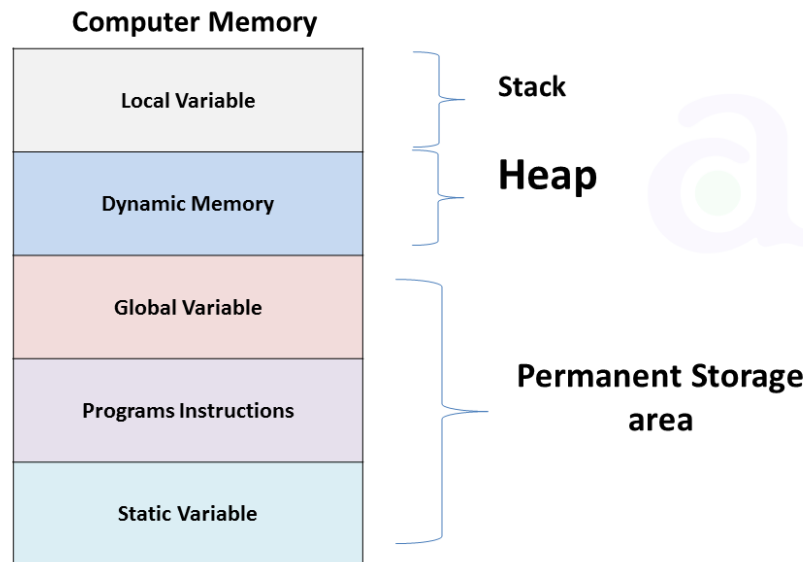
Single Indirection



Multiple Indirection

Dynamic Allocation

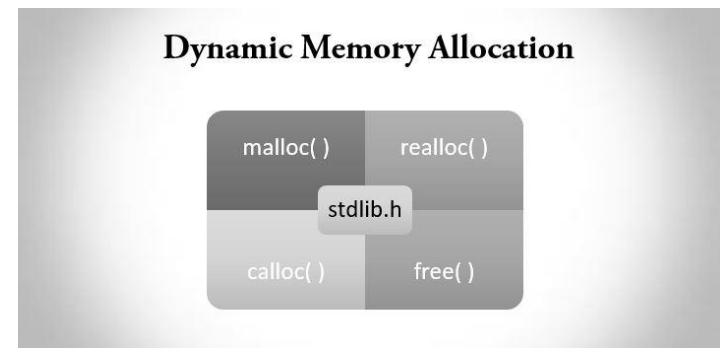
- In C, the exact size of an array is unknown until compile time.
- Sometimes the size of the array can be insufficient or more than required.
- There for Dynamic Memory allocation allows a program to obtain memory at runtime.



Cont.

- Dynamic memory allocation allows your program to obtain more memory space while running, or to release it if it's not required.
- Dynamic Allocation Functions in C,
 - **malloc()** is used to allocate memory.
 - **calloc()** allocates space for array elements.
 - **free()** is used to release memory.
 - **realloc()** change the size of previously allocated space.

Note: More dynamic allocation functions are available in C



Example

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

```
int main()
{
    char *mem_allocation;
    /* memory is allocated dynamically */
    mem_allocation = malloc( 20 * sizeof(char) );
    if( mem_allocation == NULL )
    {
        printf("Couldn't be able to allocate requested memory\n");
    }
    else
    {
        strcpy(mem_allocation, "Technology");
    }
    printf("Dynamically allocated memory content : %s\n", mem_allocation);
    free(mem_allocation); // frees the allocated memory
}
```

Output

Dynamically allocated memory content :
Technology

Comments about Pointers

- Pointers must be initialized. A common error is shown below.

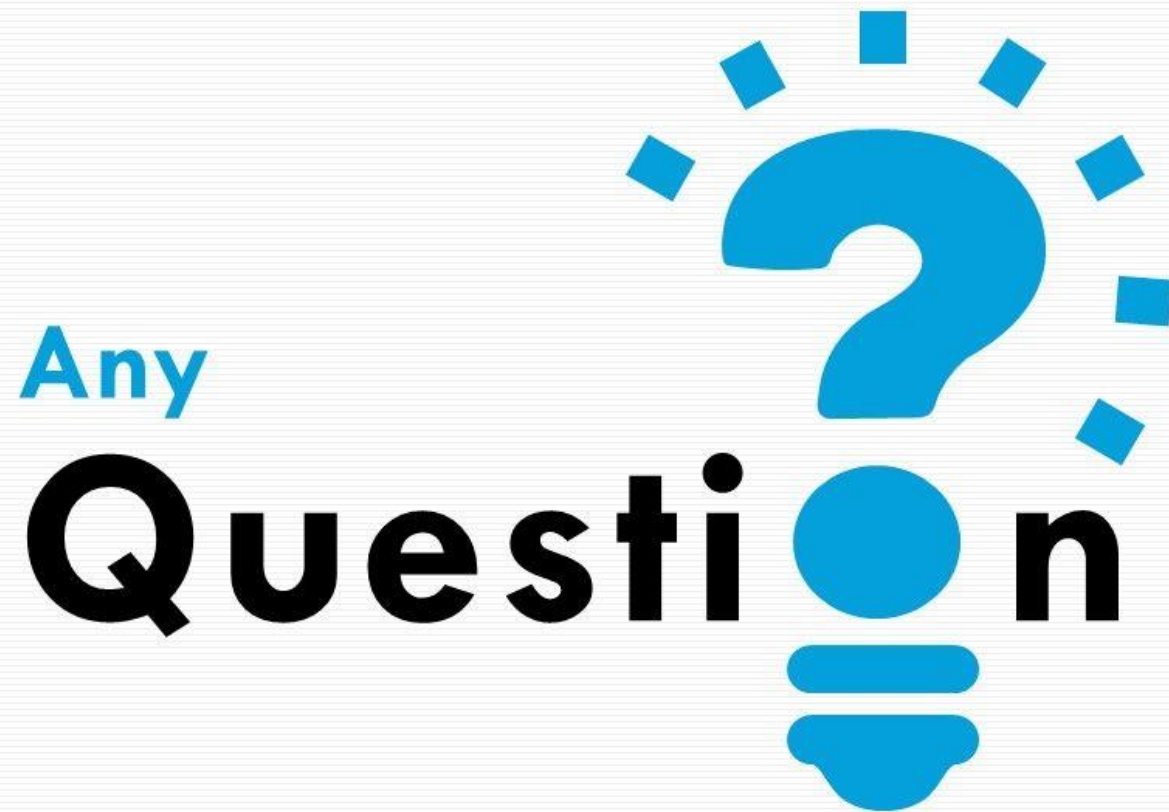
`//This program is wrong`

```
int main()
{
    int x, *p; //p can point to any memory location

    x = 10;

    *p = x;    //The value of x will be written to some
    return 0;  //                                unknown location
}
```

- Lack of understanding of pointers can lead to serious errors.



THANK YOU... !

