**Programming Fundamentals**

|  |  |
| --- | --- |
| Lab 10 | |
| **Topic** | Dynamic Memory |
| **Objective** | * DYNAMIC MEMEORY ALLOCATION   + Explaining what is Dynamic Memory Allocation.   + Explaining the problem that is solved by Dynamic Memory Allocation (referred to at the start of Pointer Topic).   + Explaining new and delete (both the forms).   + Discuss stack memory and heap memory.   + Explain the difference between creating/deleting a memory location and multiple memory location (array). |

**Lab Description:**

This lab is basically designed for the dynamic memory. We will see the use of dynamic memory and difference between stack and heap. We will see how to allocate a dynamic memory and deallocate it.

**What is Memory Allocation?**

As we discussed earlier when we need a memory for any purpose we need to allocate it before using it. There are two ways via which memories can be allocated. The two ways are:

**Compile time allocation or static allocation of memory:**

Where the memory for named variables is allocated by the compiler. Exact size and storage must be known at compile time and for array declaration, the size has to be constant.

**Runtime allocation or dynamic allocation of memory:**

Where the memory is allocated at runtime and the allocation of memory space is done dynamically within the program run, the exact space or number of the item does not have to be known by the compiler in advance. Pointers play a major role in this case.

The key word **new** is used to allocate dynamic or run time memory.

**Example:**

int \* p=nullptr, size=5;  //declares a pointer p

p = new int;   //dynamically allocate an int and loading the base address in pointer p.

p = new int[size]; //dynamically allocate an int array of size 5 and loading the base address in pointer p.

**Delete Operator:**

Memory de-allocation is also a part of this concept where the “clean-up” of space is done for variables or other data storage. It is the job of the programmer to de-allocate dynamically created space. For de-allocating dynamic memory, we use the delete operator. In other words, dynamic memory Allocation refers to performing memory management for dynamic memory allocation manually.

**Example:**

int \* p=nullptr, size=5;  //declares a pointer p

p = new int;   //dynamically allocate an int and loading the base address in pointer p.

delete p; // de allocate or free the memory.

p = new int[size]; //dynamically allocate an int array of size 5 and loading the base address in pointer p.

delete [] p; // de allocate or free the memory.

**Stack and Heap Memory:**

Memory is divided into different parts and each part is designed to perform specific tasks.

**Heap Memory**:

It is reserved for dynamic memory allocation

* In C++, when we use “new” keyword, system reserves memory in heap
* It is necessary for a programmer to free this memory when he doesn’t need it.
* To free this memory, “delete” keyword is used.

**Stack Memory:**

* Compiler calculates the size of memory needed for local variables.
* Local variables declared in a program are stored in stack.
* Memory reserved in Stack, cannot be increased or decreased during execution of a program.
* Every function has its own stack frame
* Stack frames build on one another. Like a pile of files.
* To access the file at the bottom, files from the top needs to be removed.
* Functions are like files.

**Dynamic Memory Allocation:**

Dynamic Memory Allocation is a process in which we allocate or deallocate a block of memory during the run-time of a program. There are four functions

**Lab Tasks**

Task 1:

Write the value of the variables in the boxes provided

Suppose,

X - Address is = Your Roll number e.g. BSCS2096

Y - Address is = Your Roll number (digit) + Your Roll number (digit) e.g. 2096 + 2096

= BSCS2192

Every number consider as 4B(byte).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | P | \*p | Q | \*q | X | &x | y | &y |
| void main(){ |  |  |  |  |  |  |  |  |
| int \*p = NULL; |  |  |  |  |  |  |  |  |
| int x = 10,y=20; |  |  |  |  |  |  |  |  |
| int \*q = NULL; |  |  |  |  |  |  |  |  |
| p = &x; |  |  |  |  |  |  |  |  |
| (\*p)++; |  |  |  |  |  |  |  |  |
| q = &y; |  |  |  |  |  |  |  |  |
| x = 5 + y; |  |  |  |  |  |  |  |  |
| \*p = \*p+2; |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| y = y + \*q; |  |  |  |  |  |  |  |  |
| p = q; |  |  |  |  |  |  |  |  |
| y++; |  |  |  |  |  |  |  |  |
| p++; |  |  |  |  |  |  |  |  |
| \*p; |  |  |  |  |  |  |  |  |
| q = q+3; |  |  |  |  |  |  |  |  |
| \*q;  } |  |  |  |  |  |  |  |  |

Task 2:

Write the value of the variables in the boxes provided

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Numbers | Numbers[i] | p | \*p |
| main(){ |  |  |  |  |
| int numbers[5]; |  |  |  |  |
| int\* p; |  |  |  |  |
| p = numbers; |  |  |  |  |
| \*p = 10; |  |  |  |  |
| p++; |  |  |  |  |
| \*p = 20; |  |  |  |  |
| p =&numbers[2]; |  |  |  |  |
| \*p = 30; |  |  |  |  |
| p = numbers +3; |  |  |  |  |
| \*p = 40; |  |  |  |  |
| p = numbers; |  |  |  |  |
| \*(p+4) = 50;  } |  |  |  |  |

Task 3

Determine the output of following program:

|  |
| --- |
| #include<stdio.h>  int main()  {  int a[] = {5, 15, 34, 54, 14, 2, 52, 72};  int \*p=&a[1], \*q=&a[5];  cout<<\*(p+3)<<endl;  cout<<\*(q-3)<<endl;  cout<<(int)(q-p)<<endl;  if(p<q)  cout<<"address” <<p<< value "<<\*p<<endl;  if(\*q<\*p)  cout<<\*q<<endl;  return 0;  } |

Task 4:

Write the function ***int search( int \* a, int n, int key)*** which takes an integer array or a pointer ***a*** of size ***n*** and returns 1 if integer ***key*** exists in array ***a*** and returns 0 otherwise. Write appropriate main function to test your code.

**Task 5:**

Implement following function:

* Char \*Substr(char \*str1, int pos, int length); //returns a copied portion of str1
* bool isEmpty();
* Int Find(char \*c1, char \*c2); //returns the first occurrence of c2 in c1
* Bool Compare(char \*c1, char\*c2);

**Task 6:**

Write a function ***palindrome*** which takes a character pointer ***s*** as parameter and return 1 if ***s***is a palindrome and 0 otherwise. A string is palindrome only if it one can read it the same way in both directions i.e. from left to right and from right to left. For example the following string is a palindrome: **neveroddoreven**

Write appropriate main function to test your code.

*Note: you can only use pointer notation to solve this exercise. You can use strlen library function to compute the length of* ***s***

Task 7:

Write a program to perform the following subtasks:

1. Declare an integer variable **a,** and assign **50** to it.
2. Declare a pointer to integer **ptr**.
3. Store the address of variable **a** to the pointer **ptr**. [Hint: Address operator]
4. Add **5** in the variable by using pointer **ptr**. [Hint: Indirection operator]
5. Display the value of variable by using pointer **ptr**. [Hint: Indirection operator]
6. Display the contents of pointer **ptr**.
7. Display the address of pointer **ptr**.

Task 8:

Write a Program that declares and initializes an integer array. Use pointer arithmetic to display the contents of the array. Now reverse the array contents so that first element becomes last element as shown in figure. At the end display the contents of the reversed array (only by using pointers).

**Original Array**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | 9 | 2 | ……………………. | 7 | 23 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 23  **Reversed Array** | 7 | ……………………. | 2 | 9 | 4 |