C++ Programming – Lecture 4

Arrays

- Array is a variable capable of holding > 1 value at a time
- Two basic properties of an array :
  1) Similarity - All array elements are similar to one another
  2) Adjacency - All array elements are stored in adjacent memory locations
- 2 ways to declare an array :

```
int arr[ 10 ] ;    /* mentioning size is compulsory */
int num[ ] = { 23, 34, 54, 22, 33 } ;    /* size is optional */
```

- Array elements are always counted from 0 onwards. So arr[9] is 10th element
- Array elements can be scanned OR calculated :

```
Cin >> arr[ 7 ] >> arr[ 8 ] >> arr[ 9 ] ;
arr[ 5 ] = 3 + 7 % 2 ;
```

- Arithmetic on array elements is allowed :

```
arr[ 6 ] = arr[ 1 ] + arr[ 3 ] / 16 ;
```

- Typical way to process an array element by element :

```
int arr[ 10 ] ;
for ( i = 0 ; i <= 9 ; i++ )
    /* process arr[ i ] */
```

- To obtain address of 0th element of array use :

```
int arr[ 10 ] ;  int *p ;
p = arr ;           /* method 1 */
p = &arr[ 0 ] ;   /* method 2 */
```

- On incrementing a pointer it always points to the next location of its type

  On incrementing a float pointer it points to the next float which is 4 bytes away
  On incrementing an int pointer it points to the next int which is 4 bytes away
  On incrementing a char pointer it points to the next char which is 1 byte away

- Only legal pointer operations :

  pointer + number $\rightarrow$ pointer
  pointer - number $\rightarrow$ pointer
  pointer - Pointer $\rightarrow$ number
  pointer == pointer

- 5 ways to access array elements using pointers :
  - Set up a pointer holding base address of the array :

```
int arr[ 10 ], *p ;
p = arr ;
```

- In a for loop use one of the five expressions :
  *p ; p++ ;   OR
  * (p + i)   OR   * (i + p)   OR
  p[i]   OR   i [p]

- To pass an array to a function we must always pass two things :
  1) Base address of the array
  2) Size of the array
- Array can neither grow nor shrink in size during execution of the program
- We can declare an array using int arr[n] and then receive the value of n from keyboard

Multi-dimensional Arrays

- 2-D array is a collection of several 1-D arrays
- If 2-D arrays is initialized at the same place where it is declared, then mentioning the column dimension is optional
- A 2-D array is laid out linearly in memory in row-major fashion i.e. row after row
- int *p[4] ;   - p is an array of 4 integer pointers. Size of p = 16 bytes
- int (*p)[4] ;   - p is a pointer to an array of 4 integers. Size of p = 4 bytes
- Typical applications of 2-D arrays :

  All matrix and determinant operations
- Applications of 2-D arrays in games :

  Chess, Ludo, Snakes and Ladders, Brainvita, Any other board game
- 3-D array is a collection of several 2-D arrays
- Size of a 3-D array is sum of sizes of all its elements

Strings

- To deal with strings C++ has a ready-made class called string. This class internally uses a char array.
- Useful string functions :

  front( ) - returns first character in string
  back( ) - returns last character in string
  substr( ) – returns a substring
  length( ) – returns length of a string
  append( ) – concatenates one string at end of another
  find( ) – searches a string within a string
  replace( ) – replaces a substring in a string with another string

erase( ) – deletes specified substring in a string
clear( ) – erases the entire string