Object Oriented Programming using C++

Arrays and Strings

- An array is a collection of variables of the same type that are accessed using a common name. A specific element within the array is retrieved by an index.

- Arrays are stored in contiguous memory locations.

- The first element corresponds to the lowest memory address, while the last element corresponds to the highest address.

- Arrays can have one or multiple dimensions.

Single-Dimension Arrays

- A list of items that share the same data type and are accessed using a single variable name with one subscript.

- The general form for declaring a single-dimensional array is: `type var_name[size];`

`type`: The data type of the array elements.

`var_name`: A user-defined name for the array.

- Like other variables, arrays must be explicitly declared.

Example: `int list[10];` or `char str[20];`

Single-Dimension Array initialization

- Array Initialization can occur at two stages:

1. At compile time.
2. At runtime.

Compile-time initialization:

type arr_name[size] = {list of items};

- Example:
  - `int no[3];`
  - `char str[] = {'g', 'o', 'o', 'd', '\0'};`
  - `int list[5] = {9, 18}; // Remaining elements set to zero`
  - `char str[] = "SIT";`
  - `int list[3] = {10, 11, 12, 13}; // Illegal, too many initializers`

Run-time initialization:

- Runtime Initialization is typically done using loop statements.

Example: Initializing an array called `list` with 5 numbers:

```
for (int i = 0; i < 5; i++) {
        scanf("%d", &list[i]);
    }
```

Generating pointer to an array

- It is possible to refer to an array by simply specifying the array name without any index.

For example:

```
int simple[10];
 int *ptr;
 ptr = simple;  // This is equivalent to:
 ptr = &simple[0];  // Address of the first item in the array
```

- Using `simple` or `&simple[0]` both produce the same result.

Null terminated strings

- A string is a sequence of characters treated as a single data item.

- Any group of characters defined between double quotation marks (excluding the double quotes) is a string constant.

  - Example:

- It is a null (`'\0'`) terminated character array.

- Sometimes, null-terminated strings are also referred to as C-Strings.

  - Example: `char str[12];`

  - Here, `str` has space to store the null character at the end of the string.

  - When storing "hello there" in `str`, the compiler automatically appends the null character.

- String manipulation functions are provided in the `string.h` header file.

The most common string functions are:

1. **strcpy(s1, s2);** – Copies the content of `s2` into `s1`.

2. **strcat(s1, s2);** – Concatenates `s2` to the end of `s1`.

3. **strlen(s1);** – Returns the length of the string `s1`.

4. **strcmp(s1, s2);** – Cmpares two strings, returning `+1`, `-1`, or `0`.

5. **strchr(s1, ch);** – Returns a pointer to the first occurrence of the character `ch` in `s1`.

6. **strstr(s1, s2);** – Returns a pointer to the first occurrence of `s2` in `s1`.

Passing single-dimension array to the functions

The syntax for passing an array to a function is:

```
function_name(array_name[, size]); // [size is optional]
```

For example: sum(a, 5);

The passed array can be received by the formal parameter in three ways:

1. As a sized array: int sum(int a[5])
2. As a pointer: int sum(int *a, int n)
3. As an unsized array: int sum(int a[])

Two-Dimensional array

Arrays that have elements with two subscripts are known as 2-D arrays. A 2-D array consists of rows and columns, where each element is accessed using two subscripts.

The general form of declaring a two-dimensional array is:

- type var_name[row_size][col_size];

For example, if you declare a 2-D array as `float matrix[3][6];`, it reserves 72 bytes of storage locations and can store 18 elements (3 rows * 6 columns).

The individual elements are accessed like this:

matrix[0][0], matrix[0][1], ..., matrix[2][5];

2D Array Initialization

A 2-D array can be initialized by listing the values enclosed in curly braces. There are different methods for initializing the elements of a 2-D array:

1. Manual Initialization:

```
int mat[2][2];
mat[0][0] = 1; mat[0][1] = 2;
mat[1][0] = 3; mat[1][1] = 4;
```

2. Initialization with Nested Braces:

```
int mat[2][2] = { {1, 2}, {3, 4} };
```

3. Implicit Size Initialization:

```
int mat[][2] = { {1, 2}, {3, 4} };
```

Here, the elements are specified explicitly, so there is no need to mention the row size.

If values are missing during initialization, they are automatically set to zero. For example:

```
int mat[3][5] = {1};
```

In this case, the first element of all rows will be 1, and the rest will be zeros.

Runtime initialization can be done using loops. Here's an example:

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 5; j++) {
        // Initialization code here
    }
}
```

In this loop, `i` represents the row and `j` represents the column.

Multi-Dimensional array

Arrays with more than two dimensions are also possible. The general format is:

    type array-name[size1][size2][size3]...;

For example, `int lamps[3][3][4];` defines a three-dimensional array where:

`[3]` represents the types of lamps,

`[3]` represents the wattage types,

`[4]` represents the years.

In this case, the array represents the sales data for 3 types of lamps with 3 different wattages over 4 years. The total number of integer elements in this array is `3 * 3 * 4 = 36`.