Arrays

- Array of fixed size
- Can be of any type
- Represent lists and tables
- Multidimensional arrays
- Subscripts start at 0 and end one less than the array size
- Strings are arrays of characters
  - Contained within double quotes
  - Terminated with the null character '\0'
  - String literals have \0 automatically appended
- Passed by reference to functions (pointer)

Array Declaration

- `type name[size];`

```
int n[10] = {0};    // Array of 10 integers initialized to 0

int n[5] = {32, 27, 64, 18, 95, 14};  // What is the error

// Size will be the number of elements
int n[] = {27, 64, 18, 95, 14};

// Initialize the array with a loop and algorithm
for (size_t j = 0; j < SIZE; j++)
{ s[j] = 2 + 2 * j; }
```

Single and Multi-dimensional Arrays

```
int listofnumbers[50];  // Subscripts 0 to 49
int nums[5];

thirdnumber=listofnumbers[2];  // Saving 3rd element
listofnumbers[5]=100;   // Assign 100 to the 6th element

int tableofnumbers[50][50];

anumber=tableofnumbers[2][3];
tableofnumbers[25][16]=100;
```

Iterating Through an Array

- Use a for loop for iterating through an array

```
for (size_t i = 0; i < sizeof(nums); ++i)
{ printf("%5u%10d\n", i, nums[i]); }

int total = 0;

for (size_t j = 0; j < SIZE; j++)
{ total += a[j]; }
```

Strings - array of characters

- C does not handle strings as a data type

```
char name[50];
char s[] = "Hello";
char firstname[50],lastname[50],fullname[100];

char string1[50] = "DAVE";
char string2[] = "Hello";
char string3[] = ['L','i','a','m','\0'];

printf("Enter your name (49 or less): ");
scanf("%49s",name);
```

Strings Handled by strings.h

- Strings are handled by the `string.h` library
- Many functions are available for manipulating strings

```
printf("%c\n",string1[3]);          // Prints V
printf("%s\n",string1);             // Prints DAVE

strlen(string1);                    // Returns 4
strcat(string1,string2);            // string1 = DAVEHello
strcpy(string1,string3);            // string1 = Liam
strcmp(string1,string2);            // Returns 0 if equal
```

Array Manipulation

- Arrays are passed to functions as pointers
- Copying arrays is inefficient and should be avoided
- Pointers will be covered in a later lecture
- Sorting arrays : ascending, descending
- Data Analysis : mean, median, mode, standard deviation
- Searching arrays : linear, binary

Multidimensional Array

```
for (size_t row = 0; row < 5; row++)
{
    for (size_t column = 0; column < 5; column++)
    {
        total = += a[row][column];
    }
}

// Roll a six-sided die
for (unsigned int roll = 1; roll <= 36000; ++roll)
{   // An integer from 1 to 6
    unsigned int face = 1 + rand() % 6;
    ++frequency[face];
}
```

```
// Bubble Sort
for (size_t pass = 0; pass < SIZE - 1; ++pass)
{   // Loop to control number of comparisons per pass
    for (size_t j = 0; j < SIZE - 1; ++j)
    {
        if (a[j] > a[j + 1])
        {   // Swap elements
            int hold = a[j];
            a[j] = a[j + 1];
            a[j + 1] = hold;
        }
    }
}
```

Linear Search

- Works well for small unsorted arrays
- Compare at least half of the elements

```
// Linear Search
for (size_t n = 0; n < SIZE; n++)
{
    if array[n] == key)
    {   // If the element is found
        return n;   // Return the location
    }
}
return -1;  // Element not found
```

Binary Search

- Works well for large sorted arrays

```
int low = 0;    // Low end of the search area
int high = SIZE - 1;    // High end of the search area
while (low < = high)
{
    int middle = (low + high) / 2;  // Middle element
    if (key == array[middle])
    {   // If the element is found
        return middle;  // Return the location
    }
    else if (key < array[middle])
    {   // Middle element is too high
        high = middle - 1;  // Eliminate the higher half
    }
    else {   // Middle element is too low
        low = middle + 1;   // Eliminate the lower half
    }
}
return -1;  // Element not found
```

Questions