Module 4.3:
Pointers

Pointers

- Introduction to pointers: Pointer declaration and initialization, Pointer addition and subtraction, Evaluating pointer expressions
- Pointers and Functions: Pass by Reference, Returning pointers from functions

Pointers

- Address in C/C++:
- If you have a variable var in your program, &var will give you its address in the memory.

```cpp
#include <iostream>
using namespace std;

int main() {
    int var = 5;

    cout << "var: " << var << endl;

    cout << "address of var: " << &var << endl;

    return 0;
}
```

```
var: 5
address of var: 0x61fe4c
```

Pointers

- A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location.
- Pointers (pointer variables) are special variables that are used to store addresses rather than values.
- Pointer Syntax

  int *p1;
  int p2;
  int* p1, p2;

Here p1 is pointer and p2 is normal variable

Pointers

- The general form of a pointer variable declaration is −

type *var-name;

- Here, type is the pointer's base type; it must be a valid C data type and var-name is the name of the pointer variable. Take a look at some of the valid pointer declarations −

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

- The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

Pointers

- Assigning addresses to Pointers

```
int* pc, c;
c = 5;
pc = &c;
```

Here, 5 is assigned to the c variable. And, the address of c is assigned to the pc pointer.

Why Pointer Type Matters in C++

The pointer's type determines:

- **How many bytes to read/write** when dereferencing
- **How pointer arithmetic behaves**
  - e.g., p + 1 moves by sizeof(type)
- **Which operations are allowed**
  - You cannot assign a double* to an int* without casting.

Get Value of Variable Pointed to by a Pointer

- To get the value stored at the address held by a pointer, we use the **dereference operator (*)**.

```cpp
#include <iostream>
using namespace std;

int main() {
    int* pc;  // pointer to int
    int c = 5;

    pc = &c;  // store the address of c in pointer pc

    cout << "Value of c using pointer: " << *pc << endl;

    return 0;
}
```

```
Value of c using pointer: 5
```

```cpp
#include <iostream>
using namespace std;

int main() {
    int* pc;    // pointer to int
    int c;

    c = 22;
    cout << "Address of c: " << &c << endl;
    cout << "Value of c: " << c << endl;    // 22

    pc = &c;    // pc now points to c
    cout << "\nAddress stored in pc (address of c): " << pc << endl;
    cout << "Value pointed to by pc: " << *pc << endl;    // 22

    c = 11;    // change value of c
    cout << "\nAddress stored in pc (same as before): " << pc << endl;
    cout << "Value pointed to by pc: " << *pc << endl;    // 11

    *pc = 2;    // modify c through pointer
    cout << "\nAddress of c: " << &c << endl;
    cout << "Value of c after *pc = 2: " << c << endl;    // 2

    return 0;
}
```

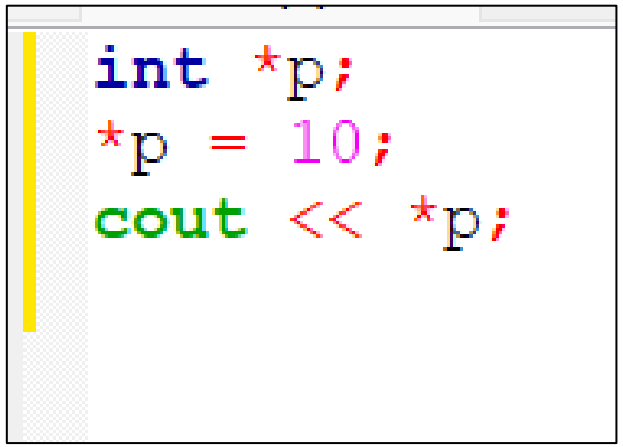
```
Address of c: 0x61fe44
Value of c: 22

Address stored in pc (address of c): 0x61fe44
Value pointed to by pc: 22

Address stored in pc (same as before): 0x61fe44
Value pointed to by pc: 11

Address of c: 0x61fe44
Value of c after *pc = 2: 2
```

Identify the error



Identify the error

```
int *p;
*p = 10;
cout << *p;
```

- p is uninitialized. It is pointing to garbage. Must assign memory first.

Pointers • Common mistakes when working with pointers

```
int c, *pc;

// pc is address but c is not
pc = c; // Error

// &c is address but *pc is not
*pc = &c; // Error

// both &c and pc are addresses
pc = &c;

// both c and *pc values
*pc = c;
```

Pointers • NULL Pointers

It is always a good practice to assign a **nullptr** value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned nullptr is called a null pointer.

```cpp
#include <iostream>
using namespace std;

int main() {
    int* ptr = nullptr;    // C++ way of creating a null pointer

    cout << "The value of ptr is: " << ptr << endl;

    return 0;
}
```

```
The value of ptr is: 0
```

Functions

Function Calls, Passing Arguments to a Function by Value

1) Function – Call by value method – In the call by value method the actual arguments are copied to the formal arguments, hence any operation performed by function on arguments doesn't affect actual parameters.

2) Function – Call by reference method – Unlike call by value, in this method, address of actual arguments (or parameters) is passed to the formal parameters, which means any operation performed on formal parameters affects the value of actual parameters.

Swap 2 numbers using pointers

```cpp
// Function to swap using pointers (Call by Reference through addresses)
void swapnum(int* var1, int* var2) {
    int tempnum = *var1;
    *var1 = *var2;
    *var2 = tempnum;
}

int main() {
    int num1 = 35, num2 = 45;

    cout << "Before swapping:\n";
    cout << "num1 value is " << num1 << endl;
    cout << "num2 value is " << num2 << endl;

    // Calling swap function by passing addresses
    swapnum(&num1, &num2);

    cout << "\nAfter swapping:\n";
    cout << "num1 value is " << num1 << endl;
    cout << "num2 value is " << num2 << endl;

    return 0;
}
```

```
Before swapping:
num1 value is 35
num2 value is 45

After swapping:
num1 value is 45
num2 value is 35
```

C++ version

- C++ also allows **reference variables** so you don't need pointers

```
void swapnum(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}
```

Pointer arithmetic (addition / subtraction)

```cpp
#include <iostream>
using namespace std;

int main() {
    int arr[] = {10, 20, 30, 40};
    int *p = arr;        // arr decays to pointer to first element
    cout << "*p = " << *p << "\n";          // 10
    cout << "*(p+1) = " << *(p+1) << "\n";// 20

    p += 2;              // moves forward by 2 * sizeof(int)
    cout << "*p after p+=2 = " << *p << "\n"; // 30

    // pointer subtraction
    int *a = &arr[3];
    int *b = &arr[0];
    cout << "a - b = " << (a - b) << "\n"; // 3
}
```

```
*p = 10
*(p+1) = 20
*p after p+=2 = 30
a - b = 3
```

Pre-increment and Post-increment of Pointers

- Pointer increment does **not** add 1 to the pointer value.
  Instead, it moves the pointer to the **next memory location of its base type**.

- Example:
  If int is 4 bytes and p is an int*, then:
- p + 1 moves ahead by **4 bytes**
- p++ and ++p also move the pointer by **4 bytes**

Post-increment: p++

```cpp
#include <iostream>
using namespace std;

int main() {
    int arr[] = {10, 20, 30};
    int* p = arr;

    cout << *p << endl;     // 10
    cout << *p++ << endl;   // prints 10, then p moves to next element
    cout << *p << endl;     // now p points to 20
}
```

- *p++= *(p++)= use value at current p i.e 10 then increment pointer so it now points to 20

Pre-increment: ++p

```
int arr[] = {10, 20, 30};
int* p = arr;

cout << *++p << endl;    // increments p, then dereferences
```

- ++p = increment pointer first .
- now points to 20
  Then * prints the value = 20

Pointers and arrays

int num[] = {23, 34, 12, 44, 56, 17};

Memory layout

Element	Value	Address
num[0]	23	4000
num[1]	34	4004
num[2]	12	4008
num[3]	44	4012
num[4]	56	4016
num[5]	17	4020

Accessing array elements using pointers

```cpp
#include<iostream>
using namespace std;

int main()
{


int num[] = {24, 34, 12, 44, 56, 17};

int* p = num;    // base address
while (p <= num + 5) {
    cout << "address = " << p;
    cout << " element = " << *p<<endl;
    p++;         // move to next element
}
return 0;
}
```

```
address = 0x61fe30 element = 24
address = 0x61fe34 element = 34
address = 0x61fe38 element = 12
address = 0x61fe3c element = 44
address = 0x61fe40 element = 56
address = 0x61fe44 element = 17
```

Passing arrays to functions

Int num[10];

The array name (e.g., num) automatically converts to a pointer num and is equivalent to &num[0]

display(num, 6); // name of array = address of 0th element

```cpp
#include <iostream>
using namespace std;

void display(int* p, int n) {
    int i = 0;
    while (i < n) {
        cout << "Element = " << *p << endl;
        p++;        // move pointer to next element
        i++;
    }
}

int main() {
    int num[] = {24, 34, 12, 44, 56, 17};

    // passing address of first element
    display(num, 6);        // same as display(&num[0], 6)

    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

int main() {
    int num[] = {24, 34, 12, 44, 56, 17};

    for (int i = 0; i < 6; i++) {
        cout << "address = " << &num[i] << "   ";

        cout << "num[i] = " << num[i] << "   ";
        cout << "*(num + i) = " << *(num + i) << "   ";


        cout << endl;
    }

    return 0;
}
```

```
address = 0x61fe30   num[i] = 24   *(num + i) = 24
address = 0x61fe34   num[i] = 34   *(num + i) = 34
address = 0x61fe38   num[i] = 12   *(num + i) = 12
address = 0x61fe3c   num[i] = 44   *(num + i) = 44
address = 0x61fe40   num[i] = 56   *(num + i) = 56
address = 0x61fe44   num[i] = 17   *(num + i) = 17
```

num[i] is the same as *(num + i)

Try

Write a C program that performs the following operations on an array using functions and pointers:

1. Accept N numbers from the user and store them in an array using a function.
2. Use a pointer to traverse the array and find both the largest and smallest elements.
3. Implement a function to search for a given number in the array and return all indices where the number appears.
4. Display the largest, smallest elements, and indices of the searched number (if found).

Implement following functions: ***accept(), largest(), smallest(), search()***

```cpp
void accept(int* arr, int n);
int largest(int* arr, int n);
int smallest(int* arr, int n);
int search(int* arr, int n, int key, int* indices);
int main() {
    int n, key;

    cout << "Enter number of elements: ";
    cin >> n;
    int arr[n];       // dynamic array
    int indices[n];  // to store positions where key is found
    accept(arr, n);
    int maxVal = largest(arr, n);
    int minVal = smallest(arr, n);
    cout << "\nEnter number to search: ";
    cin >> key;

    int count = search(arr, n, key, indices);
    cout << "Largest element : " << maxVal << endl;
    cout << "Smallest element: " << minVal << endl;

    if (count > 0) {
        cout << "Number " << key << " found at indices: ";
        for (int i = 0; i < count; i++) {
            cout << indices[i] << " ";
        }
        cout << endl;
    } else {
        cout << "Number " << key << " NOT found in the array.\n";
    }

    return 0;
}
```

```
Enter number of elements: 5

Enter 5 numbers:
1
5
3
8
8

Enter number to search: 8

===== Results =====
Largest element : 8
Smallest element: 1
Number 8 found at indices: 3 4
```

```cpp
void accept(int* arr, int n) {
    cout << "\nEnter " << n << " numbers:\n";
    for (int i = 0; i < n; i++) {
        cin >> *(arr + i);  // using pointer arithmetic
    }
}
int largest(int* arr, int n) {
    int maxVal = *arr;      // first element
    for (int i = 1; i < n; i++) {
        if (*(arr + i) > maxVal)
            maxVal = *(arr + i);
    }
    return maxVal;
}

int smallest(int* arr, int n) {
    int minVal = *arr;
    for (int i = 1; i < n; i++) {
        if (*(arr + i) < minVal)
            minVal = *(arr + i);
    }
    return minVal;
}
int search(int* arr, int n, int key, int* indices) {
    int count = 0;

    for (int i = 0; i < n; i++) {
        if (*(arr + i) == key) {
            indices[count] = i; // store the index
            count++;
        }
    }
    return count; // number of times key found
```

Try

Write a C Program that uses pointers & functions to copy contact of two input strings str1 & str2 into a str3 using user defined function. (Do not use Inbuilt string functions)
Function: concatenateStrings(str1, str2, str3)

Enter first string: Hello Enter second string: World Concatenated String: HelloWorld	Enter first string: Ram Enter second string: Sita Concatenated String: RamSita

```
void concatenateStrings(char* str1, char* str2, char* str3) {
    // Copy str1 into str3 using pointers
    while (*str1 != '\0') {
        *str3 = *str1;  // copy character
        str3++;
        str1++;
    }

    // Append str2 to str3
    while (*str2 != '\0') {
        *str3 = *str2;
        str3++;
        str2++;
    }

    *str3 = '\0';   // null terminate the final string
}

int main() {
    char str1[100], str2[100], str3[200];

    cout << "Enter first string: ";
    cin >> str1;

    cout << "Enter second string: ";
    cin >> str2;

    concatenateStrings(str1, str2, str3);
```

```
Enter first string: ABC
Enter second string: PQR
Concatenated String: ABCPQR
```

Exercise

Develop a **C program** using pointers and functions to perform geometric computations on a **square**. Your program should:

1. **Compute the distance** between two points (x1,y1) and (x2,y2) using a function that takes pointers as arguments.
2. **Compute the area** of a square given its four vertices (x1,y1), (x2,y2), (x3,y3), and (x4,y4) using the distance function.
3. **Determine if a given point** (x,y) lies inside the square by checking coordinate constraints.

Imaplent below functions: ***distance() , area(), inside()***

Predict the output?

```
int arr[] = {10, 20, 30, 40};
int *p = arr;

cout << *(p + 2) << " ";
cout << p[3] << " ";
cout << 3[p];
```

Identify the error?

```
int* fun() {
    int x = 10;
    return &x;
}

int main() {
    int *p = fun();
    cout << *p;
}
```

What happened if we pass update(a)?

```
void update(int *x) {
    *x = *x + 5;
}

int main() {
    int a = 10;
    update(&a);
    cout << a;
}
```

What is the output?

```
int a = 5;
int b = 5;
if (&a == &b) cout << "same";
else cout << "different";
```

Predict the output

```
int arr[] = {1,2,3,4};
int *p = arr + 1;
cout << *(p++) << " " << *p;
```

Output?

```
int a[3] = {1, 2, 3};
cout << *a + *(a+1) + a[2];
```

Output?

```
int a = 5;
int b = 10;
int *p = &b;
p = &a;
cout << *p;
```

Output?

```
int a = 10;
int b = 20;
int *p1 = &a;
int *p2 = &b;
cout << (p2 - p1);
```

Ans: Undefined (pointer arithmetic allowed only within same array)