Examples on Pointers:

```
#include <iostream>
using namespace std;
int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int* ptr = arr; // Pointer initialized to point to the first element of the array
    // Accessing array elements using the pointer
    for (int i = 0; i < 5; i++) {
    cout << "Value at position " << i << " is: " << *ptr <<endl;
        ptr++; // Move the pointer to the next element
    }
    // Resetting the pointer to the beginning of the array
    ptr = arr;
    cout << "Value at current pointer: " << *ptr << endl;
    cout << "Value at next element (ptr + 1): " << *(ptr + 1) << endl;
    cout << "Value at third element (ptr + 2): " << *(ptr + 2) << endl;
```

```
    // Moving pointer backward and forward
    ptr += 3; // Move the pointer 3 positions ahead
    cout << "\nAfter moving pointer ahead by 3 positions, value: " << *ptr << endl;
    ptr -= 2; // Move the pointer 2 positions back
    cout << "After moving pointer back by 2 positions, value: " << *ptr << endl;
    return 0;}
```

The Out Put

```
Value at position 0 is: 10
Value at position 1 is: 20
Value at position 2 is: 30
Value at position 3 is: 40
Value at position 4 is: 50

Value at current pointer: 10
Value at next element (ptr + 1): 20
Value at third element (ptr + 2): 30

After moving pointer ahead by 3 positions, value: 40
After moving pointer back by 2 positions, value: 20
```

Traversing a String Using Pointer

```
#include <iostream>
using namespace std;
int main() {
    char str[] = "Hello, Pointer!";
    char* ptr = str; // Pointer to the first character of the string
    cout << "Traversing string using pointer:" << endl;
    while (*ptr != '\0') { // Stop at null terminator
        cout << *ptr;      // Print the character the pointer points to
        ptr++;             // Move to the next character
    }  return 0;}
```

Out Put

Traversing string using pointer:

Hello, Pointer!

Pointer with Multiple Data Types

```
#include <iostream>
using namespace std;
int main() {
    int intArr[] = {1, 2, 3, 4, 5};
    double doubleArr[] = {1.1, 2.2, 3.3, 4.4, 5.5};
    int* intPtr = intArr;
    double* doublePtr = doubleArr;
    cout << "Integer array traversal using pointer:" << endl;
    for (int i = 0; i < 5; i++) {
        cout << "intArr[" << i << "] = " << *intPtr << endl;
        intPtr++;
    }
    cout << "\nDouble array traversal using pointer:" <<
endl;
    for (int i = 0; i < 5; i++) {
        cout << "doubleArr[" << i << "] = " << *doublePtr <<
endl;
        doublePtr++;
    }
    return 0;
}
```

Out Put

Integer array traversal using pointer:

intArr[0] = 1

intArr[1] = 2

intArr[2] = 3

intArr[3] = 4

intArr[4] = 5

Double array traversal using pointer:

doubleArr[0] = 1.1

doubleArr[1] = 2.2

doubleArr[2] = 3.3

doubleArr[3] = 4.4

doubleArr[4] = 5.5

Difference Between Pointers

```
#include <iostream>
using namespace std;
int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int* start = arr;        // Pointer to the first element
    int* end = arr + 4;      // Pointer to the last element
    cout << "Pointer Difference Example:" << endl;
    cout << "Distance between start and end: " << (end
- start) << endl;
    cout << "Value at start: " << *start << endl;
    cout << "Value at end: " << *end << endl;
    return 0;}
```

Out Put

```
Pointer Difference Example:
Distance between start and end: 4
Value at start: 10
Value at end: 50
```

Swapping Two Variables Using Pointers

```
#include <iostream>
using namespace std;
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main() {
    int x = 10, y = 20;
    cout << "Before Swap: x = " << x << ", y = " << y << endl;
    swap(&x, &y);
    cout << "After Swap: x = " << x << ", y = " << y << endl;
    return 0;}
```

Output:

Before Swap: x = 10, y = 20

After Swap: x = 20, y = 10

Summing Array Elements Using Pointers

```
#include <iostream>
using namespace std;
int main() {
    int arr[] = {5, 10, 15, 20, 25};
    int* ptr = arr;
    int sum = 0;
    for (int i = 0; i < 5; i++) {
        sum += *ptr; // Add value at current pointer
        ptr++;       // Move to next element
    }
    cout << "Sum of array elements: " << sum << endl;
    return 0;
}
```

Output:

```
Sum of array elements: 75
```

Reversing an Array Using Pointers

```
#include <iostream>
using namespace std;
void reverseArray(int* start, int* end) {
    while (start < end) {
        int temp = *start;
        *start = *end;
        *end = temp;
        start++;
        end--;
    }
}
int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = 5;
    cout << "Original Array: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    reverseArray(arr, arr + size - 1);
```

```
    cout << "\nReversed Array: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }    return 0;
}
```

Output:

Original Array: 1 2 3 4 5

Reversed Array: 5 4 3 2 1

لماذا نستخدم المؤشرات

١. **إدارة الذاكرة بشكل مباشر حيث ان المؤشرات تتيح الوصول المباشر إلى مواقع الذاكرة.**

٢. **تمرير القيم بكفاءة:** عند تمرير القيم إلى الدوال باستخدام المؤشرات، يمكننا تعديل القيم مباشرة في مواقعها الأصلية، بدلاً من إنشاء نسخة من القيمة (كما يحدث عند تمرير القيم بالنسخ).

٣. **العمل مع المصفوفات والسلاسل النصية** المؤشرات ترتبط ارتباطًا وثيقًا بالمصفوفات، حيث يمكن استخدامها للوصول إلى عناصر المصفوفة بكفاءة.

٤. **العمل مع الهياكل المعقدة** تستخدم لبناء الهياكل المعقدة مثل القوائم المرتبطة , الأشجار الثنائية والجداول (Hash Tables).

٥. **التحكم في المؤشرات واستخدام الدوال:** تستخدم المؤشرات للوصول إلى الدوال من خلال المؤشرات إلى الدوال هذا مفيد لتطوير دوال ديناميكية أو استدعاء دوال معينة بناءً على مدخلات المستخدم.

٦. **التعامل مع المؤشرات المتقدمة :** مثل المؤشرات الذكية (Smart Pointers) في C++ الحديثة التي تستخدم لإدارة الموارد بشكل آمن وفعّال دون القلق بشأن تسريبات الذاكرة

٧. **التعامل مع الواجهات منخفضة المستوى** مثل قراءة البيانات مباشرة من عناوين ذاكرة محددة وكذلك التعامل مع المؤشرات عند كتابة برامج تعمل مع النواة

٨. **دعم البرمجة الكائنية :**المؤشرات تُستخدم للوصول إلى الكائنات والدوال الأعضاء بشكل ديناميكي