|  |  |
| --- | --- |
| week | three |
| class | SS1 and 2 |
| Lesson title | Introduction to pointers in c |
| subtitle | The concept of c pointers |
| period |  |
| duration | 40 mins |
| Learning/instructional objective | At the end of the lesson, learners should be able to:  Explain how a machine switches values in memory  Explain the significance of pointers in memory management |
| date |  |
| Key vocabulary | pointer |
| Resources and instructional materials |  |
| Previous knowledge | Students have learnt about functions in c programming |

Introduction

Pointers are variables that store memory addresses.

They are a fundamental concept in C and are used extensively in system programming and low-level operations.

Syntax of c pointers

The syntax of pointers is similar to the variable declaration in C, but we use the ( \* ) dereferencing operator in the pointer declaration.

datatype \* ptr;

where

ptr is the name of the pointer.

datatype is the type of data it is pointing to.

The above syntax is used to define a pointer to a variable. We can also define pointers to functions, structures, etc.

How to Use Pointers?

The use of pointers in C can be divided into three steps:

Pointer Declaration

Pointer Initialization

Pointer Dereferencing

|  |  |
| --- | --- |
| week | Four |
| class | SS1 and 2 |
| Lesson title | Introduction to pointers in c |
| subtitle | Working with pointers in c |
| period |  |
| duration | 40 mins |
| Learning/instructional objective | At the end of the lesson, learners should be able to:  Declare pointers  Assign values to pointers  Access values stored in a pointer |
| date |  |
| Key vocabulary | pointer |
| Resources and instructional materials |  |
| Previous knowledge | Students have learnt about the basic concept of pointers |

**Declaring Pointers**

Pointers are declared with a specific data type to indicate the type of data they point to.

The \* symbol is used to declare a pointer variable. For example:

int \*ptr; // Declares a pointer to an integer

**Accessing and Assigning Values**

To access the value pointed to by a pointer, you use the dereference operator \*.

To assign a value to the location pointed to by a pointer, you use the dereference operator on the left side of an assignment.

int x = 10;

int \*ptr = &x; // Pointer points to the address of x

int y = \*ptr; // y now holds the value 10

\*ptr = 20; // x is now 20

|  |  |
| --- | --- |
| week | Five (live coding session) |
| class | SS1 and 2 |
| Lesson title | Introduction to pointers in c |
| subtitle | Working with pointers in c |
| period |  |
| duration | 40 mins |
| Learning/instructional objective | At the end of the lesson, learners should be able to:  Declare pointers  Assign values to pointers  Access values stored in a pointer |
| date |  |
| Key vocabulary | pointer |
| Resources and instructional materials |  |
| Previous knowledge | Students have learnt about the basic concept of pointers |

1. Write a c program to add two numbers
2. Write a c program to switch two numbers

Solution

1. A c program to add two numbers using variables

#include <stdio.h>

int result1(int test,int exam){

int result = test + exam;

return result;

}

int main(){

int test = 2;

int exam = 4;

int result = result1(test,exam);

printf("%d\n", result);

return 0;

}

Limitations with this approach

When the values of test and exam are passed to the result1 function, new copies (duplicates) of the values are created in memory, hence causing a wastage of memory. In environments where memory is a scarce commodity, this can be a very serious challenge for instance in embedded systems programming etc.

#include <stdio.h>

int add(int \*a, int \*b) {

return (\*a + \*b);

}

// int add1(int a, int b){

// return a+ b;

// }

int main(void) {

int x = 10, y = 20;

int result;

int result1 = 0;

printf("before adddition, value of result1 = %d\n", result1);

// printf("Before adding: x = %d, y = %d\n", x, y);

// result = add(&x, &y);

result1 = add1(x,y);

printf("after adddition, the new value of result1 is: %d\n", result1);

// printf("Result of addition: %d\n", result1);

return 0;

}

/\*

The add function takes two pointers to integers, adds their values, and returns the result. In the main function, we call the add function to add the values of x and y, store the result in a variable result, and then print the result.

\*/

In the second approach, the same problem is solved, but now using a different approach. Instead of variables, pointers are used to access the values in memory. This resolves to the same value being used multiple times, hence preventing memory wastage.

|  |  |
| --- | --- |
| week | nine |
| class | SS1 and 2 |
| Lesson title | Introduction to pointers in c |
| subtitle | Working with pointers in c (2) |
| period |  |
| duration | 40 mins |
| Learning/instructional objective | At the end of the lesson, learners should be able to:  Use pointers to carry out arithemetic operations  Explain the concept of null pointers  Use a pointer to point to a pointer |
| date |  |
| Key vocabulary | pointer |
| Resources and instructional materials |  |
| Previous knowledge | Students have learnt about the basic concept of pointers |

**Pointer Arithmetic**

Pointers can be used in arithmetic operations. This is particularly useful for iterating through arrays.

Adding an integer to a pointer increments (or decrements) it by a multiple of the size of the data type it points to.

int arr[] = {1, 2, 3, 4, 5};

int \*ptr = arr; // Points to the first element

ptr++; // Moves to the next element

**NULL Pointers**

A pointer that doesn't point to a valid memory location is often set to NULL.

Accessing or manipulating a NULL pointer can lead to undefined behavior.

int \*ptr = NULL;

if (ptr == NULL) {

// Handle NULL pointer

}

**Pointer to Pointers**

Pointers can point to other pointers, creating multi-level pointer structures.

Useful when working with arrays of pointers or dynamic memory allocation.

int x = 10;

int \*ptr1 = &x;

int \*\*ptr2 = &ptr1; // ptr2 is a pointer to a pointer to an int

|  |  |
| --- | --- |
| week | ten |
| class | SS1 and 2 |
| Lesson title | Introduction to pointers in c |
| subtitle | Working with pointers in c (3) |
| period |  |
| duration | 40 mins |
| Learning/instructional objective | At the end of the lesson, learners should be able to:  Work with arrays using pointers  Use pointers on functions  Explain the concept of dynamic memory allocation |
| date |  |
| Key vocabulary | pointer |
| Resources and instructional materials |  |
| Previous knowledge | Students have learnt about the basic concept of pointers |

**Arrays and Pointers**

Arrays are closely related to pointers in C.

The name of an array is a pointer to its first element.

You can use array notation or pointer notation interchangeably.

int arr[] = {1, 2, 3, 4, 5};

int \*ptr = arr; // ptr points to the first element

int val = arr[2]; // Accessing array element using array notation

int val2 = ptr[2]; // Accessing using pointer notation

**Pointer to Functions**

Pointers can also point to functions.

Useful for implementing function pointers and callback mechanisms.

int add(int a, int b) {

return a + b;

}

int (\*functionPtr)(int, int) = add; // Pointer to a function

int result = functionPtr(2, 3); // Calls the add function

**Dynamic Memory Allocation**

Pointers are commonly used to manage dynamically allocated memory using functions like malloc, calloc, and realloc.

Don't forget to release memory using free to avoid memory leaks.

int \*dynArray = (int\*)malloc(5 \* sizeof(int)); // Allocate memory for an array

free(dynArray); // Release allocated memory when done

Pointers and Structures

Pointers can be used to work with structures.

Arrow -> operator is used to access structure members through a pointer to a structure.

struct Point {

int x;

int y;

};

struct Point p1;

struct Point \*ptr = &p1;

ptr->x = 10; // Accessing structure member using a pointer

**Common Pitfalls**

Avoid using uninitialized pointers.

Be cautious of pointer arithmetic to prevent memory access violations.

Always check for NULL pointers before dereferencing.

Conclusion

Pointers are a powerful feature in C, but they come with a responsibility to manage memory and avoid common pitfalls. They are essential for tasks like dynamic memory allocation, data manipulation, and working with complex data structures.

Understanding and mastering pointers is crucial for C programming, and it can greatly enhance your ability to control and optimize your code.