EXPERIMENT8: POINTERS

Experiment 1: Declaration and Initialization of Pointers

Aim

To declare and initialize pointers of different data types and display their addresses and values.

Algorithm

1. Declare int, float, and char variables
2. Declare pointers for each data type
3. Assign addresses of variables to pointers
4. Print variable values, pointer values, and addresses

Pseudocode

```
DECLARE int a, float b, char c
DECLARE int*, float*, char*
ASSIGN addresses of variables to pointers
PRINT variable values
```

```
PRINT pointer values and dereferenced values
```

Flowchart (ASCII)

```
Start
   ↓
Declare Variables
   ↓
Declare Pointers
   ↓
Assign Addresses
   ↓
Display Values
   ↓
 End
```

C Program

```c
#include <stdio.h>

int main() {
    int a = 10;
    float b = 3.14;
    char c = 'A';

    int *p1 = &a;
    float *p2 = &b;
    char *p3 = &c;

    printf("Integer value: %d, Address: %p\n", a, p1);
    printf("Float value: %.2f, Address: %p\n", b, p2);
    printf("Char value: %c, Address: %p\n", c, p3);
```

```
    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc point1.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
Integer value: 10, Address: 0061FF10
Float value: 3.14, Address: 0061FF0C
Char value: A, Address: 0061FF0B
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

Pointers successfully store addresses of variables and allow indirect access to their values.

Experiment 2: Pointer Arithmetic

Aim

To perform increment and decrement operations on pointers and observe memory address changes.

Algorithm

1. Declare integer array
2. Assign base address to pointer
3. Increment and decrement pointer
4. Display addresses and values

Pseudocode

```
DECLARE array
DECLARE pointer
POINT to array[0]
INCREMENT pointer
DECREMENT pointer
DISPLAY addresses and values
```

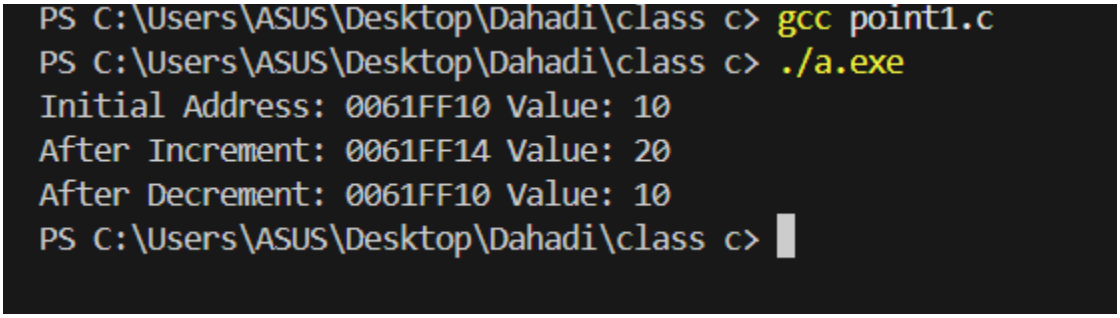
Flowchart (ASCII)

```
Start
   ↓
Declare Array
   ↓
Assign Pointer
   ↓
Increment / Decrement Pointer
   ↓
Display Values
   ↓
 End
```

C Program

```c
#include <stdio.h>

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

    printf("Initial Address: %p Value: %d\n", p, *p);

    p++;
    printf("After Increment: %p Value: %d\n", p, *p);

    p--;
    printf("After Decrement: %p Value: %d\n", p, *p);

    return 0;
}
```

Output



Conclusion

Pointer arithmetic changes the address based on the size of the data type (e.g., 4 bytes for int).

Experiment 3: Passing Variables by Reference Using Pointers

Aim

To modify variable values by passing them to a function using pointers.

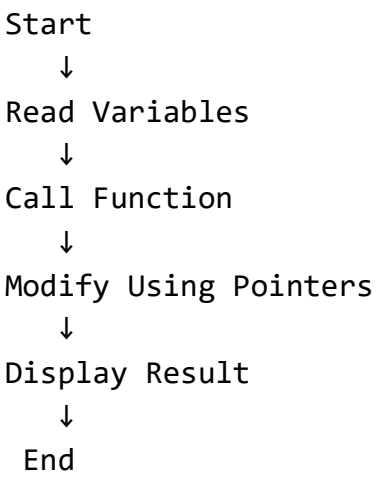
Algorithm

1. Declare variables
2. Pass their addresses to a function
3. Modify values inside the function
4. Display modified values

Pseudocode

```
FUNCTION modify(x, y)
    *x = *x + 10
    *y = *y * 2
END FUNCTION

MAIN
    PASS addresses to function
    DISPLAY modified values
```

Flowchart (ASCII)



C Program

```c
#include <stdio.h>

void modify(int *x, int *y) {
    *x = *x + 10;
    *y = *y * 2;
}

int main() {
    int a = 5, b = 4;

    printf("Before: a=%d b=%d\n", a, b);
    modify(&a, &b);
    printf("After: a=%d b=%d\n", a, b);

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc point1.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
Before: a=5 b=4
After: a=15 b=8
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

Passing variables by reference using pointers allows functions to modify original values.