

# Indirect Access

Name: jasveena

Student ID: 224001588

Date: 1 -06 -2024

## Learning Journey and Evidence

### Review pointers, and references.

What did you think while reviewing this in the [Programmer's Guide](#)? Which aspects did you already know, which were challenging? Which explanations helped, which were confusing? Etc.

Pointers: A pointer is a data type built into the programming language. A pointer has a value, that stores the location of another value. The pointer's value is the memory address of the value it points to this means that we can point to any value in memory, regardless of where it is. A pointer value is the same as any other value it can be stored in local variables, global variables, it can be passed to a function in parameter and it can be returned from a function.

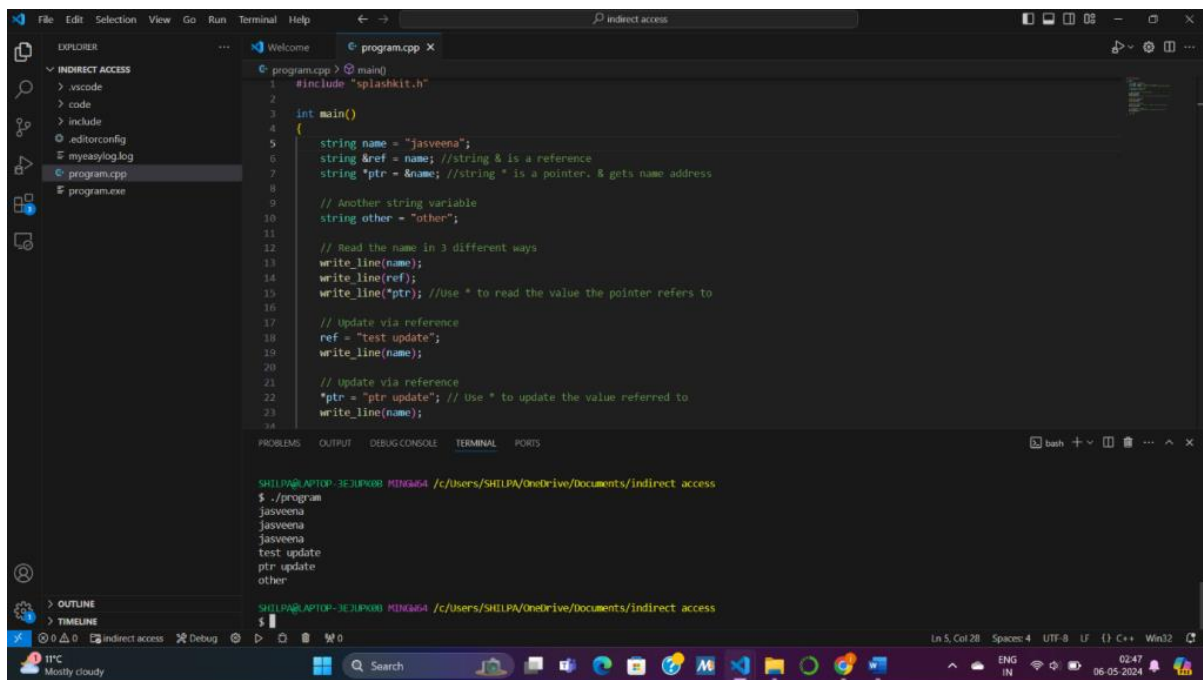
References:. Pointers and references both store the address of another value in memory. They are both used to refer to another value. While in pointer we need to manually get the address and store it in the pointer in reference the compiler takes care of this for us.

What other resources did you use to help understand these concepts? Any good resources you would recommend to others?

I used youtube to deepen my understanding of different headers

### Create a small program to demonstrate the use of pointers.

Show your process for this and highlight any realisations you gain.



The screenshot displays the Visual Studio Code interface. The Explorer panel on the left shows a project named 'INDIRECT ACCESS' with files like '.vscode', 'code', 'include', 'editorconfig', 'myeasylog.log', 'program.cpp', and 'program.exe'. The main editor window shows the source code of 'program.cpp' with the following content:

```
1 #include "splashkit.h"
2
3 int main()
4 {
5     string name = "jasveena";
6     string &ref = name; //string & is a reference
7     string *ptr = &name; //string * is a pointer. & gets name address
8
9     // Another string variable
10    string other = "other";
11
12    // Read the name in 3 different ways
13    write_line(name);
14    write_line(ref);
15    write_line(*ptr); //Use * to read the value the pointer refers to
16
17    // Update via reference
18    ref = "test update";
19    write_line(name);
20
21    // Update via reference
22    *ptr = "ptr update"; // Use * to update the value referred to
23    write_line(name);
24 }
```

The Output panel at the bottom shows the execution results of the program:

```
SHELLPA@LAPTOP-3E3UPK08I MINAKASA /c/Users/SHELLPA/OneDrive/Documents/indirect access
$ ./program
jasveena
jasveena
jasveena
test update
ptr update
other
SHELLPA@LAPTOP-3E3UPK08I MINAKASA /c/Users/SHELLPA/OneDrive/Documents/indirect access
$
```

The status bar at the bottom indicates the file is 'Ln 5, Col 28', uses 'UTF-8' encoding, and is a C++ file in the 'Win32' architecture.

Include a hand execution of the program to explore how it works.

**HAND EXECUTION (PROGRAM-1)**

**VARIABLES**

- NAME  $\Rightarrow$  (initialized with the value "Jasveena")
- REF  $\Rightarrow$  a reference ('&') to the 'name' variable.
- PTR  $\Rightarrow$  a pointer (\*), it holds the memory address of 'name'.
- OTHER  $\Rightarrow$  it is another 'String Variable'.

**OUTPUTS**

write\_line (name);  $\Rightarrow$  outputs what is said in "name"  
i.e. Jasveena

write\_line (ref);  $\Rightarrow$  same as (name)

write\_line (\*ptr);  $\Rightarrow$  same as (name).

**Terminal**

```
Jasveena
Jasveena
Jasveena
```

output  $\Rightarrow$  test update

**UPDATE VIA REFERENCE**

```
ref = "test update";
write_line (name);
```

**UPDATE VIA POINTER (\*PTR)**

```
*ptr = "ptr update";
write_line (name);
```

**CHANGE POINTER REFERENCE**

```
ptr = &other;
write_line (*ptr);
```

**Final Terminal**

```
Jasveena
Jasveena
Jasveena
Test update
ptr update
other
```

Create a small program to demonstrate the use of pass-by reference.

Show your process for this and highlight any realisations you gain.

```

program.cpp
8  draw_player(const player_data &player)
9
10 const double PLAYER_SPEED = 3;
11 const double PLAYER_RADIUS = 10;
12
13 void draw_player(const player_data &player)
14 {
15     fill_circle(color_purple(), player.x, player.y, PLAYER_RADIUS);
16 }
17
18 void update_player(player_data &player)
19 {
20     if (key_down(LEFT_KEY))
21     {
22         player.x -= PLAYER_SPEED;
23     }
24     if (key_down(RIGHT_KEY))
25     {
26         player.x += PLAYER_SPEED;
27     }
28 }
29
30 int main()
31 {
32
33     // Create a player data structure
34     player_data player;
35     player.x = 100;
36     player.y = 100;
37
38     // Draw the player
39     draw_player(player);
40
41     // Update the player
42     update_player(player);
43
44     // Draw the player again
45     draw_player(player);
46
47     return 0;
48 }

```

```

SHELLPA@LAPTOP-3E3UPK00 MINGW64 /c/Users/SHELLPA/OneDrive/Documents/indirect access
$ ./program

SHELLPA@LAPTOP-3E3UPK00 MINGW64 /c/Users/SHELLPA/OneDrive/Documents/indirect access
$ clang++ program.cpp -l splashkit -o program

SHELLPA@LAPTOP-3E3UPK00 MINGW64 /c/Users/SHELLPA/OneDrive/Documents/indirect access
$ ./program

```

Include a hand execution of one of the calls to swap to explore how it works.

## Update your [Fly Catch](#) program to make use of pass by reference

Show your process for this and highlight any realisations you gain.

```

program.cpp
62 void draw_catcher(const catcher_data &catcher)
63 {
64     // Draw the catcher
65     fill_circle(color_black(), catcher.x, catcher.y, 10);
66 }
67
68 void fly_catch(fly_data &fly, catcher_data &catcher)
69 {
70     // Check if the fly is within the catcher's radius
71     double dx = fly.x - catcher.x;
72     double dy = fly.y - catcher.y;
73     double distance = sqrt(dx * dx + dy * dy);
74     if (distance <= 10)
75     {
76         // The fly has been caught
77         fly.x = -1;
78         fly.y = -1;
79     }
80 }
81
82 int main()
83 {
84     // Create a fly data structure
85     fly_data fly;
86     fly.x = 100;
87     fly.y = 100;
88
89     // Create a catcher data structure
90     catcher_data catcher;
91     catcher.x = 500;
92     catcher.y = 500;
93
94     // Draw the fly
95     draw_fly(fly);
96
97     // Draw the catcher
98     draw_catcher(catcher);
99
100    // Catch the fly
101    fly_catch(fly, catcher);
102
103    return 0;
104 }

```

```

SHELLPA@LAPTOP-3E3UPK00 MINGW64 /c/Users/SHELLPA/OneDrive/Documents/fly catch indirect access
$ ./PROGRAM

```

Capture any other study and practice used to master these concepts.

Show evidence of any additional study and practice you did to master these concepts.

## Complete one of the [Test Your Knowledge](#) activities

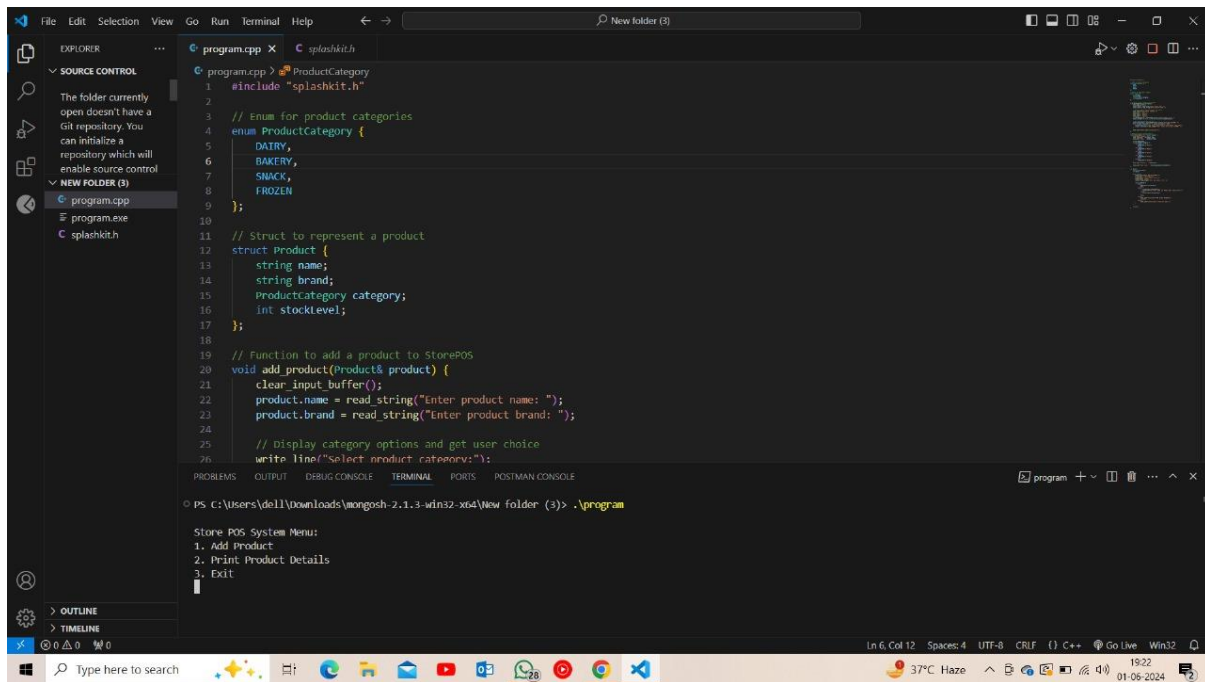
Show your process for this and highlight any realisations you gain.

The image displays two screenshots of a C++ program in Visual Studio, illustrating the execution of a 'Store POS' system.

**Top Screenshot:** The code defines a `ProductCategory` enum with values `DAIRY`, `BAKERY`, `SNACK`, and `FROZEN`. It defines a `Product` struct with fields `name`, `brand`, `category`, and `stocklevel`. A function `add_product` is implemented, which prompts the user to enter the product name and brand, and then displays the category options. The terminal output shows the message "Product added successfully!" and the "Store POS System Menu:" with options: 1. Add Product, 2. Print Product Details, 3. Exit.

**Bottom Screenshot:** The program is running, and the user has entered the following input:

```
3. Exit
1
Enter product name: Amul
Enter product brand: Amul
Select product category:
0. Dairy
1. Bakery
2. Snack
3. Frozen
```



```
File Edit Selection View Go Run Terminal Help
program.cpp X splashkit.h
SOURCE CONTROL
The folder currently open doesn't have a Git repository. You can initialize a repository which will enable source control.
NEW FOLDER (3)
program.cpp
program.exe
splashkit.h
1 #include "splashkit.h"
2
3 // Enum for product categories
4 enum ProductCategory {
5     DAIRY,
6     BAKERY,
7     SNACK,
8     FROZEN
9 };
10
11 // Struct to represent a product
12 struct Product {
13     string name;
14     string brand;
15     ProductCategory category;
16     int stocklevel;
17 };
18
19 // Function to add a product to StorePOS
20 void add_product(Product& product) {
21     clear_input_buffer();
22     product.name = read_string("Enter product name: ");
23     product.brand = read_string("Enter product brand: ");
24
25     // Display category options and get user choice
26     write_line("Select product category:");
27
28     PS C:\Users\dell\Downloads\mongodb-2.1.3-win32-x64\New folder (3)> .\program
29
30 Store POS System Menu:
31 1. Add Product
32 2. Print Product Details
33 3. Exit
34
```



## Brief Summary of Concepts

Concept	Key Idea / Concept
Pass by Value vs Pass by Reference	These are the terms that explain how data is passed to a parameter. Pass by value copies value to the parameter while pass by reference is like passing the variable itself to the parameter.
Pointers	It is a datatype built into the programming language
References	While in pointer we need to manually get the address and store it in the pointer in reference the compiler takes care of this for us.
Null	Indicates an invalid or non-binding value or associated with the value 0.
Segmentation Fault	if you attempt to dereference a pointer and perform an action that is not permitted at that location in memory, this will result in a segmentation fault.
Dangling Pointers	Dangling pointer is a pointer that does not point to a valid value.

## Reflection

### What gives you confidence you have achieved the learning goals?

*Use pass-by reference to accept and update values*

Passing parameters by reference allows you to modify the original variables passed to a function. I used the pass by reference to accept and update values in the program shown above.

*Use **const (constant)** references to provide read-only access to data to improve performance.*

Utilizing const references ensures that the data passed to a function cannot be modified within the function, which can improve performance by avoiding unnecessary copying of data. I Read through the programmers field guide and used it in the small program

*Use pointers to refer to and interact with other values in memory.*

Pointers are powerful tools for interacting with memory addresses and manipulating data indirectly. By creating programs that demonstrate the use of pointers, we'll gain a deeper understanding of how they work and their practical applications, such as dynamic memory allocation and data structures.

### What is the most important thing you learned from this and why?

The most important thing i learned from this exercise is the fundamental concepts of pointers and references and how they differ in terms of syntax, behavior, and usage. Understanding these concepts is crucial for writing efficient and robust code, especially in scenarios where memory management and performance are critical.

