Lesson:

Pointers 2

PW SKILLS

Pre-Requisites

- Basic understanding of pointers

List of Concepts Involved

- Call-by-reference using pointers
- Pointer arithmetic
- Arrays as pointers

Topic: Call-by-reference using pointers

A function can only return a single value. However, in some cases we may want it to return multiple values. In such cases, pointers come in handy. We can pass the variables that we want to be updated as pointer arguments.
For example, if we want a function to return the first position and last position of the character 'a' in a string, we can make a function of the following type-

Code
https://pastebin.com/aQLC5Gxa

Output: 0 2

In the function we are sending in the addresses of the variables first and last. These addresses are going to their respective pointer arguments. Now when we are updating the values of the dereference pointers, we are actually updating values of the objects that are stored in the respective addresses stored in them i.e. the original variables first and last. In this, you could have made the function return either the first or the last occurrence of 'a' and used the other as a pointer argument. That's just how you want to implement it. The main point here to note is how we are using pointers to get the function to return to us multiple values.
Second advantage of using pointers to pass arguments is minimizing the memory usage. When we pass an argument by value, a copy of the same is made, which in turn takes up some memory. For primitive data types like int, double etc this memory space may not be significant. However if we talk about large classes or containers(like arrays), passing by value will cause us a lot of memory. That's where the role of pointers comes into play. When we send these arguments as pointers, the amount of memory consumed is equal to the memory that a pointer takes up, which is fairly small compared to the actual size of the class or container that it points to.

For instance, if we take an integer array of 10 elements. Sending it normally would cost us

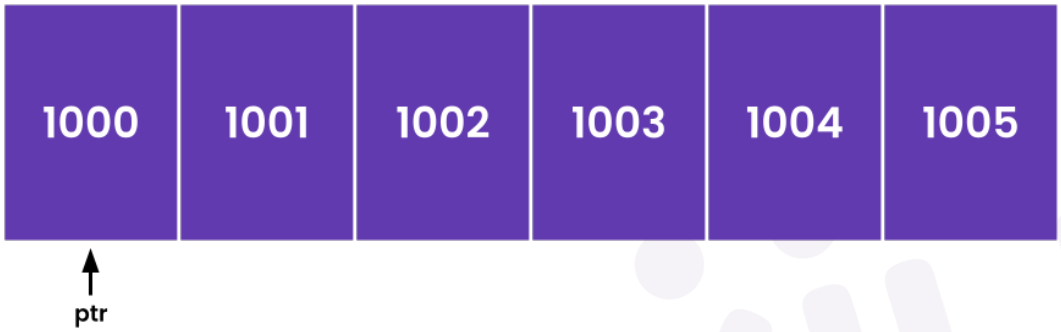
4 (size of int in bytes) * 10 = 40 bytes. However if we send the same using a pointer, the memory consumption would be only 4 bytes (the size of the integer pointer).

Topic: Pointer arithmetic

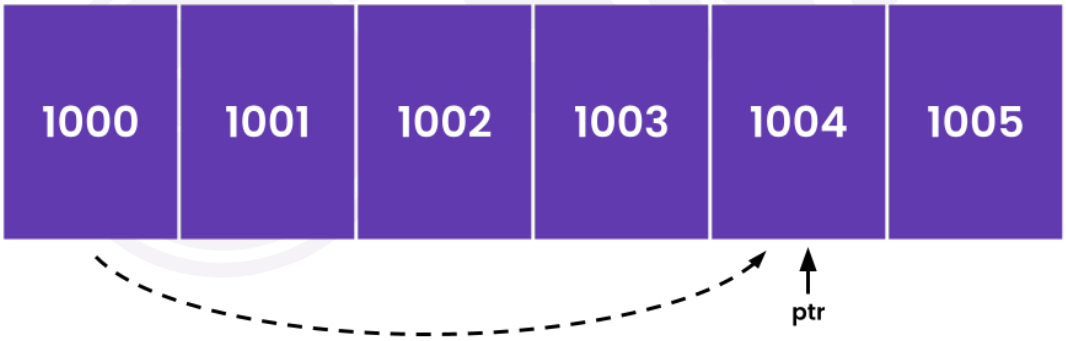
Pointers only support 2 types of operations:

1. Increment
2. Decrement

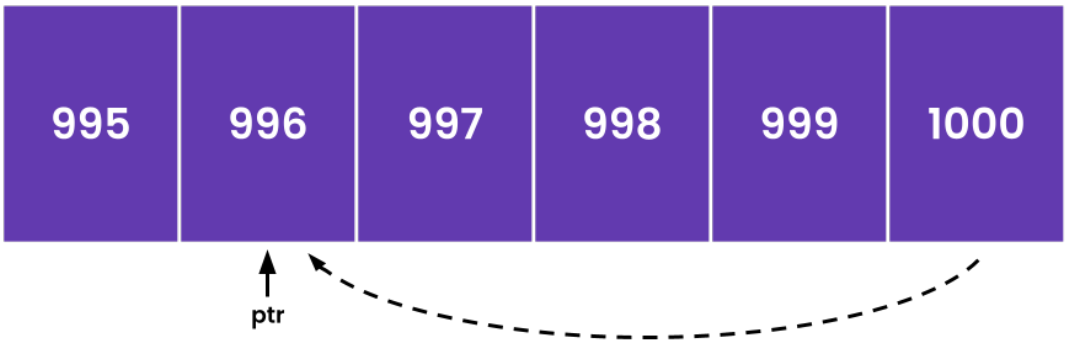
These operations are different from what we have learnt in integer arithmetic. Here increment or decrement of a pointer value refers to the shift in memory location that the pointer is pointing to. The size of the shift depends on the type of variable it points to.
For example, let's say we have a pointer variable which points to an integer object located at memory index 1000.



We know that the amount of memory that an int variable takes is 4 bytes. So, when we apply the increment operator (++) on the pointer, it moves to the memory location that is 4 bytes ahead of it.



Similarly, if we would have applied the decrement operator (--) on the pointer, it would have moved to the memory location that is 4 bytes before it.



Code
https://pastebin.com/spgUs4Yy

Output:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[2] = {1, 2};
    int *ptr = &arr[0];
    cout << "Before updation\n";
    cout << "Ptr: " << ptr << '\n';
    cout << "Value in ptr: " << *ptr << '\n';
    cout << "Array: [" << arr[0] << ", " << arr[1] << "]\n\n";

    (*ptr)++;

    cout << "After updation\n";
    cout << "Ptr: " << ptr << '\n';
    cout << "Value in ptr: " << *ptr << '\n';
    cout << "Array: [" << arr[0] << ", " << arr[1] << "]\n";

    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Coding\PW> cd "d:\Coding\PW\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
Before updation
Ptr: 0x61fef4
Value in ptr: 1
Array: [1, 2]

After updation
Ptr: 0x61fef4
Value in ptr: 2
Array: [2, 2]
```

2.`*++ptr` // increment the pointer and then dereference the address at the incremented value

Example-

Code
https://pastebin.com/A7tv8enM

Output:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[2] = {1, 2};
    int *ptr = &arr[0];
    cout << "Before updation\n";
    cout << "Ptr: " << ptr << '\n';
    cout << "Value in ptr: " << *ptr << '\n';
    cout << "Array: [" << arr[0] << ", " << arr[1] << "]\n\n";

    (*ptr)++;

    cout << "After updation\n";
    cout << "Ptr: " << ptr << '\n';
    cout << "Value in ptr: " << *ptr << '\n';
    cout << "Array: [" << arr[0] << ", " << arr[1] << "]\n";

    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Coding\PW> cd "d:\Coding\PW\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
Before updation
Ptr: 0x61fef4
Value in ptr: 1
Array: [1, 2]

After updation
Ptr: 0x61fef4
Value in ptr: 2
Array: [2, 2]
```

3.`++*ptr // increment the value the pointer points to`

Example-

Code

https://pastebin.com/NR6pALgV

Output:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[2] = {1, 2};
    int *ptr = &arr[0];
    cout << "Before updation\n";
    cout << "Ptr: " << ptr << '\n';
    cout << "Value in ptr: " << *ptr << '\n';
    cout << "Array: [" << arr[0] << ", " << arr[1] << "]\n\n";

    (*ptr)++;

    cout << "After updation\n";
    cout << "Ptr: " << ptr << '\n';
    cout << "Value in ptr: " << *ptr << '\n';
    cout << "Array: [" << arr[0] << ", " << arr[1] << "]\n";

    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Coding\PW> cd "d:\Coding\PW\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
Before updation
Ptr: 0x61fef4
Value in ptr: 1
Array: [1, 2]

After updation
Ptr: 0x61fef4
Value in ptr: 2
Array: [2, 2]
```

`4.(*ptr)++ // dereference the pointer and post-increment that value`

Example-

Code

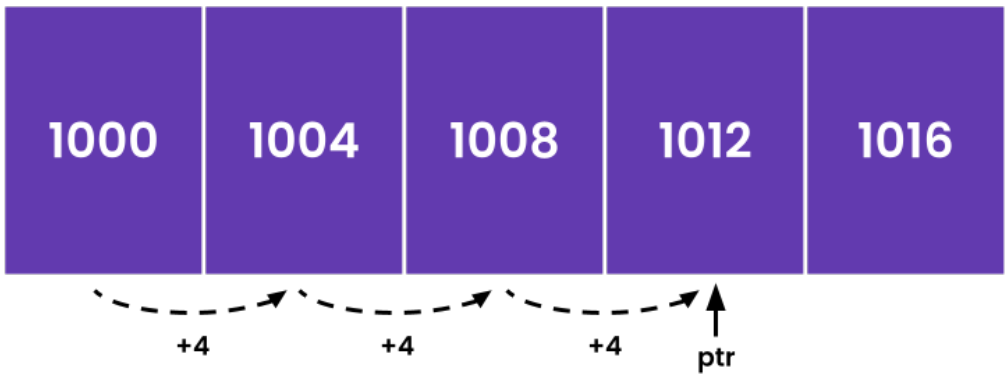
https://pastebin.com/MzHv0jEf

Output:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[2] = {1, 2};
    int *ptr = &arr[0];
    cout << "Before updation\n";
    cout << "Ptr: " << ptr << '\n';
    cout << "Value in ptr: " << *ptr << '\n';
    cout << "Array: [" << arr[0] << ", " << arr[1] << "]\n\n";

    (*ptr)++;

    cout << "After updation\n";
    cout << "Ptr: " << ptr << '\n';
    cout << "Value in ptr: " << *ptr << '\n';
    cout << "Array: [" << arr[0] << ", " << arr[1] << "]\n";

    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Coding\PW> cd "d:\Coding\PW\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
Before updation
Ptr: 0x61fef4
Value in ptr: 1
Array: [1, 2]

After updation
Ptr: 0x61fef4
Value in ptr: 2
Array: [2, 2]
```

Note that the increment or decrement doesn't have to be only of one unit. It may be larger.
For example, let's say we have an int pointer **ptr** pointing to the memory location 1000.
On this we apply the operation **ptr = ptr + 3**, as a result we'll reach the location that is 3 units ahead of the current position i.e. 1000 + 3 * 4 = 1012. (4 is the size of the int variable).



Topic: Arrays as pointers

- In C++, arrays and pointers are intertwined. In particular, we'll see that whenever we make an array, it is converted to a pointer by the compiler.
- Normally when we want to get a pointer to an object, we use the address-of operator. The same is the case with the elements of an array. We can obtain the pointer to any element of the array by taking its address.

```
int *ptr = &arr[0]
```

However, arrays have a special property. In most cases when we use an array, the compiler automatically substitutes a pointer to the first element.

```
int *ptr = arr;
```

This will automatically make ptr point to the first element of the array.

Code

https://pastebin.com/2qUQErtT

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[2] = {1, 2};
    int *ptr = &arr[0];
    cout << ptr << '\n';
    ptr = arr;
    cout << ptr << '\n';

    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Coding\PW> cd "d:\Coding\PW\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
0x61ff04
0x61ff04
```

- Pointers that address the elements of an array have an additional property- they can be used to traverse the elements of the array. For instance we can use increment operator to move from one element in an array to next,

Code

https://pastebin.com/sNw92daf

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[] = {2, 3, 4, 1};
    int *ptr = arr;
    ++ptr;
    cout << *ptr;

    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS D:\Coding\PW> cd "d:\Coding\PW\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
3
```

In this example we can see that we have initialized the pointer ptr with the address of the first element of the array. When we apply the increment operator on it, it starts pointing to the second element of the array i.e. 3. Similarly we can use the decrement operator to get access to the previous element of the array.

Code

https://pastebin.com/yM25Rtx0

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[] = {2, 3, 4, 1};
    int *ptr = arr;
    ++ptr;
    cout << *ptr;

    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS D:\Coding\PW> cd "d:\Coding\PW\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
3
```

In this ptr was initialized with the second element of the array and using the decrement operator, we made it point to the first.Note that to use a pointer to traverse through the array, it is not necessary to always initialize the pointer with the first element of the array. We can use any element of the array and then use increment and decrement operators to traverse the array.

- We can use a pointer to point to an entire array and not just one element. This is helpful specially when we are dealing with multidimensional arrays.

Syntax- **`data_type (*pointer_name)[size_of_array];`**

For example, **`int (*ptr)[10];`**

This is a pointer to an int array of size 10.

When we dereference a pointer, we get the value to which it points. In case of a pointer to an array, it refers to the base address of the array i.e. the address of the index 0 of the array.For example,

Code
https://pastebin.com/4VeEZjcc

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[3] = {1, 2, 3};
    int(*ptr)[3] = &arr;
    cout << ptr << ' ' << (*ptr);

    return 0;
}
```

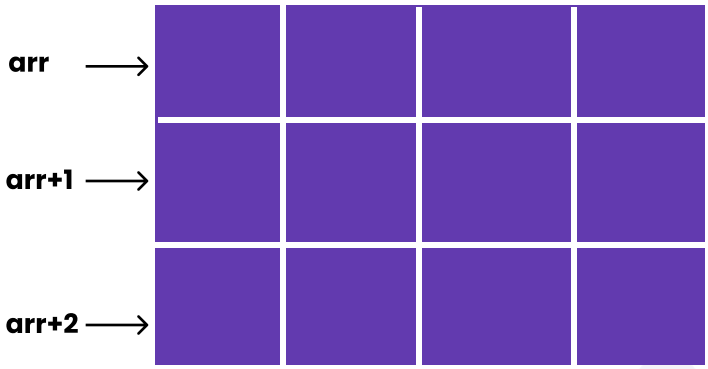
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Coding\PW> cd "d:\Coding\PW\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
0x61ff00 0x61ff00
```

Here we can clearly see that both the pointer and its deferenced value are the same, as the value stored in the pointer is the same as the address of the array.

- In case of 2D arrays, the name of the array acts as a pointer to an entire array.For example, if we have an array `arr[3][4],`

`arr` points to the first row of the matrix.

Similarly, `(arr + 1)` points to the second row and `(arr + 2)` points to the third row respectively.



Dereferencing each of these pointers will give us the respective arrays to which they point. So writing `*(arr + i)` is the same as asking for `arr[i].`

Now if we want to access individual elements of the matrix, we'll have to dereference the array pointer further. We can do this in the similar fashion as we did with the array.`*(*(arr + i) + j)`

Here we simply consider `*(arr + i)` as an individual array. So to access the jth element of this array we can simply add j to the array name and dereference it.For example if we want to access the element `arr[2][3]`, we'll write `*(*(arr + 2) + 3).`

Upcoming Class Teasers:

- Types of pointers