Pointers

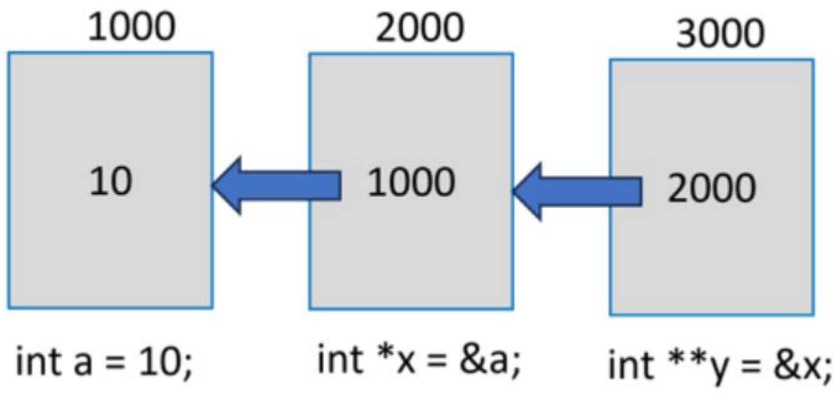
Content

- Recap
- Pointers Arithmetic
- Array Pointer

Recap

- What is Pointers?
- Declaration
- Types of Pointers
- Example of Pointers

Pointer to pointer

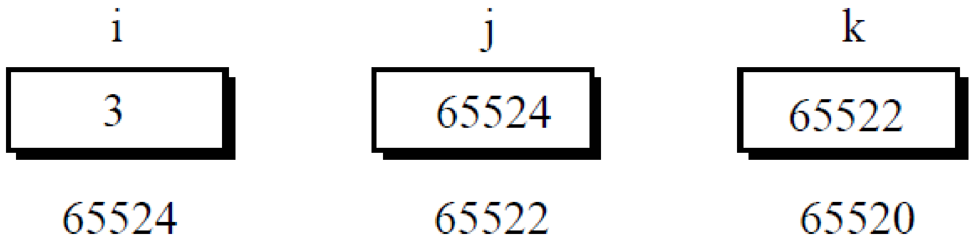


- "a" is a normal "int" variable, whose pointer is "x". In turn, the variable stores the address of "x".

- "y" is declared as "int **" to indicate that it is a pointer to another pointer variable. Obviously, "y" will return the address of "x" and "*y" is the value in "x" (which is the address of "a").

- To obtain the value of "a" from "y", we need to use the expression "**y". Usually, "y" will be called as the **pointer to a pointer**.

Example: Pointer to pointer



i j k

3 65524 65522

65524 65522 65520

```
#include<stdio.h>
int main( )
{
int i = 3, *j, **k ;
j = &i ;
k = &j ;
printf ( "\nAddress of i = %u, i = %u, i = %u ", &i, j, *k ) ;
printf ( "\nAddress of j = %u, j = %u, j = %u", &j, k, &k ) ;
printf ( "\nValue of j = %u", j ) ;
printf ( "\nValue of k = %u", k ) ;
printf ( "\nValue of i = %d, i = %d, i = %d, i = %d", i, * ( &i ), *j,  **k) ;
}
```

```
Output:
Address of i = 201259388, i = 201259388, i = 201259388
Address of j = 201259376, j = 201259376, j = 201259368
Value of j = 201259388
Value of k = 201259376
Value of i = 3, i = 3, i = 3, i = 3
```

Example: Pointer to pointer

```
#include <stdio.h>
int main(){
   int var = 10;
   int *intptr = &var;
   int **ptrptr = &intptr;
   printf("var: %d \tAddress of var: %d \n",var, &var);
   printf("inttptr: %d \tAddress of inttptr: %d \n", intptr, &intptr);
   printf("var: %d \tValue at intptr: %d \n", var, *intptr);
   printf("ptrptr: %d \tAddress of ptrtptr: %d \n", ptrptr, &ptrptr);
   printf("intptr: %d \tValue at ptrptr: %d \n\n", intptr, *ptrptr);
   printf("var: %d \t*intptr: %d \t**ptrptr: %d", var, *intptr, **ptrptr);
   return 0;}
```

Output:

```
var: 10   Address of var: 2030583852
inttptr: 2030583852        Address of inttptr: 2030583840
var: 10   Value at intptr: 10
ptrptr: 2030583840         Address of ptrtptr: 2030583832
intptr: 2030583852         Value at ptrptr: 2030583852
var: 10   *intptr: 10      **ptrptr: 10
```

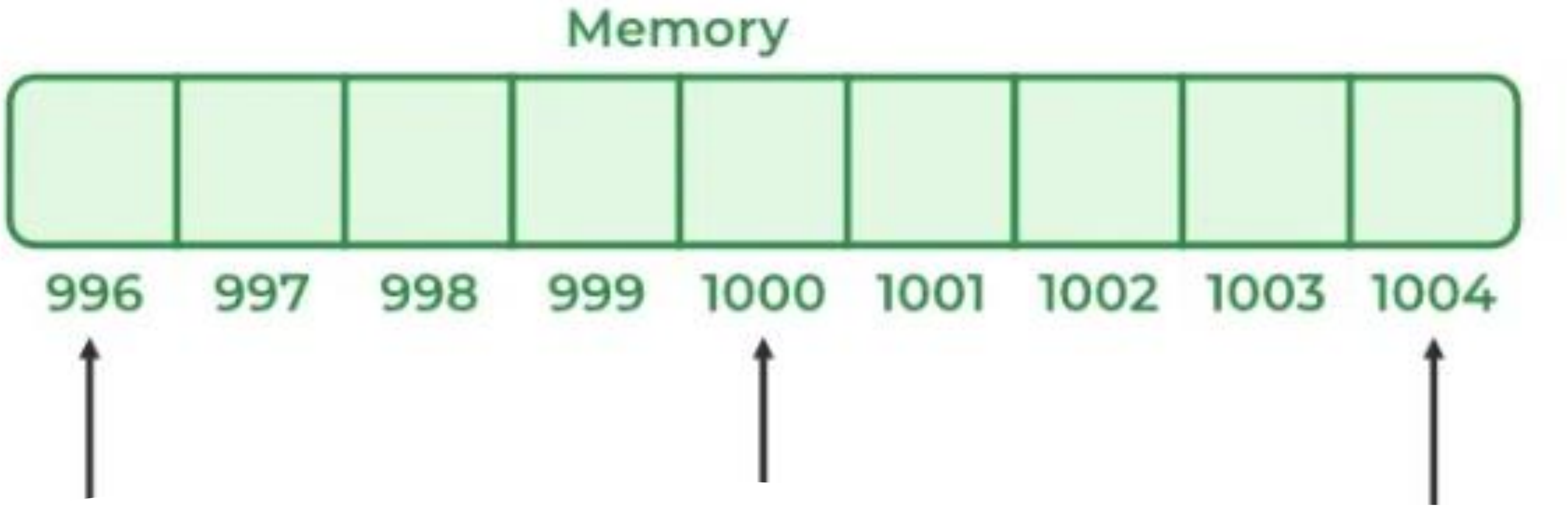
Pointer Arithmetic

A pointer variable stores the address of another variable.

The address is always an integer. So, can we perform arithmetic operations.

The following are some of the important pointer arithmetic operations in C:

- Increment and Decrement of a Pointer
- Addition and Subtraction of Integer to Pointer
- Subtraction of Pointers
- Comparison of Pointers



Increment and Decrement of a Pointer

- We know that "++" and "--" are used as the increment and decrement operators.
- They are unary operators, used in prefix or postfix manner with numeric variable operands

Increment & Decrement of a Pointer: Example

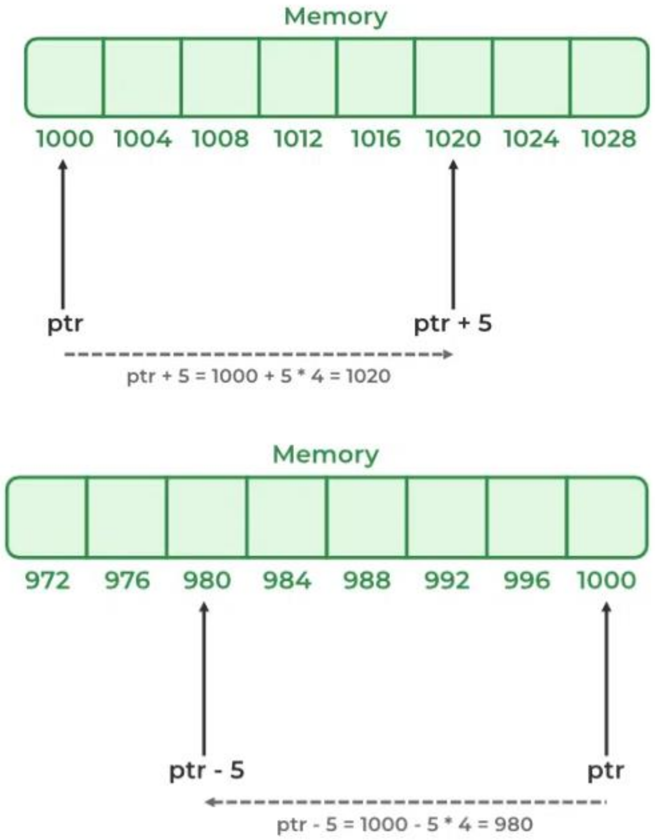
```
#include <stdio.h>

int main(){
   int x = 10;
   int *y = &x;
   printf("Value of y before increment: %d\n", y);
   y++;
   printf("Value of y after increment: %d\n", y);
   y--;
   printf("Value of y after decrement: %d", y);
}
```

Output:

```
Value of y before increment: -1556292876
Value of y after increment:    -1556292872
Value of y after decrement:   -1556292876
```

Addition and Subtraction of Integer to Pointer

```
#include <stdio.h>
int main()
{
    int N = 4;
    int *ptr;
    ptr = &N;
    printf("Pointer ptr before Addition:%p \n", ptr);
    ptr = ptr + 5;
    printf("Pointer ptr after Addition: %p \n", ptr);
    ptr = ptr - 3;
    printf("Pointer ptr after Subtraction: %p \n", ptr);
    return 0;
}
```

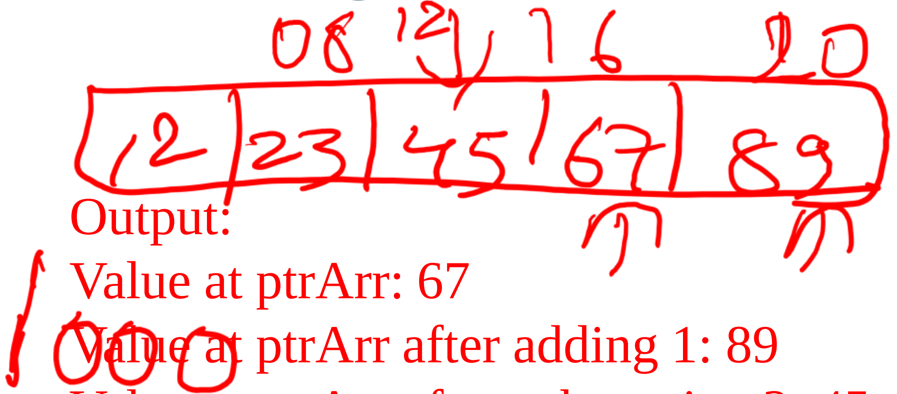


Output:

```
Pointer ptr before Addition:0x7ffff192f3bc
Pointer ptr after Addition: 0x7ffff192f3d0
Pointer ptr after Subtraction: 0x7ffff192f3c4
```

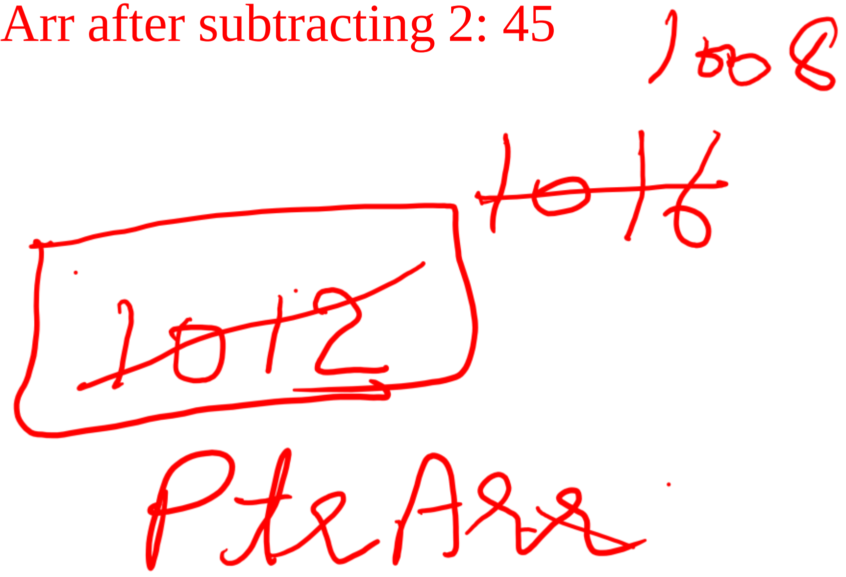
Addition and Subtraction of Integer to Pointer

```
#include <stdio.h>

int main() {
  int int_arr[] = {12, 23, 45, 67, 89};
  int *ptrArr = &int_arr[3];
  printf("Value at ptrArr: %d\n", *ptrArr);
  ptrArr = ptrArr + 1;
  printf("Value at ptrArr after adding 1: %d\n",
*ptrArr);
   ptrArr = ptrArr - 2;
 printf("Value at ptrArr after subtracting 2: %d\n",
*ptrArr);
  return 0;
}
```



Output:

```
Value at ptrArr: 67
Value at ptrArr after adding 1: 89
Value at ptrArr after subtracting 2: 45
```



Subtraction of two pointers

```
int main(){

   int a[]= {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
   int *x = &a[0]; // zeroth element
   int *y = &a[9]; // last element

   printf("Add of a[0]: %ld add of a[9]: %ld\n", x, y);
   printf("Subtraction of two pointers: %ld", y-x-5);    //When subtracting two pointers, the result
                                                         is the number of elements between them
 printf(“Addition of two pointers: %ld", y-x+5);
}
```

Output:

```
Add of a[0]: 140729350774896 add of a[9]: 140729350774932
Subtraction of two pointers: 4
Addition of two pointers: 14
```

Comparison of Pointer

```c
#include <stdio.h>
const int MAX = 3;
int main() {
  int var[] = {10, 100, 200};
  int i, *ptr1, *ptr2;
  ptr1 = var; // Initializing pointers
  ptr2 = &var[MAX - 1];
  while (ptr1 <= ptr2) {
    printf("Address of var[%d] = %p\n", i, ptr1);
    printf("Value of var[%d] = %d\n", i, *ptr1);
    ptr1++; /* point to the previous location */
    i++;
  }
  return 0;}
```

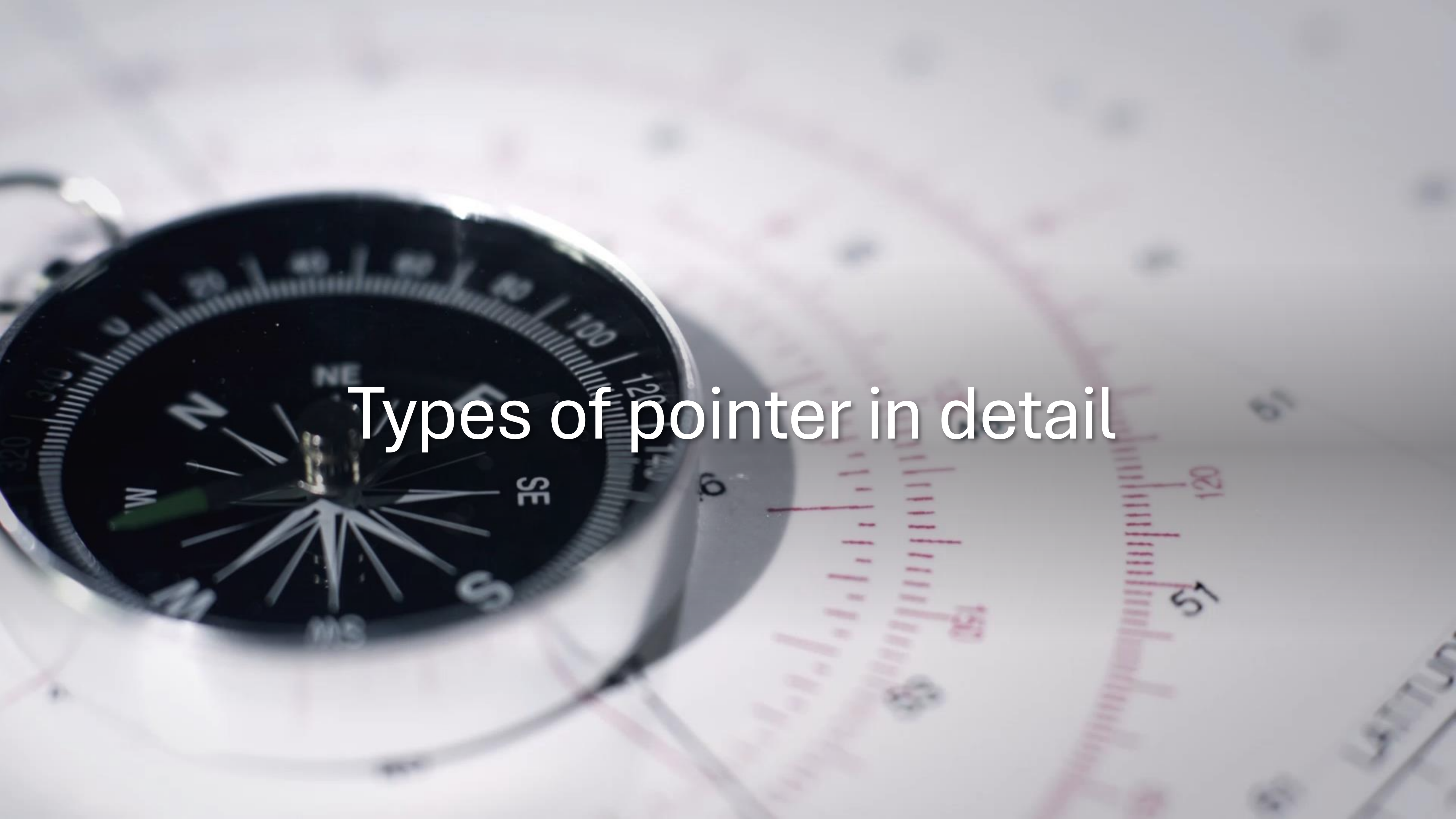
Output:

```
Address of var[0] = 0x7ffd5de2e63c
Value of var[0] = 10
Address of var[1] = 0x7ffd5de2e640
Value of var[1] = 100
Address of var[2] = 0x7ffd5de2e644
Value of var[2] = 200
```

Common pointer Mistakes

- **Uninitialized Pointers:** Using pointers without assigning a valid address.
- **Dangling Pointers**: Pointers that refer to a memory location that has been freed.
- **Pointer Arithmetic Errors**: Incorrect pointer increment/decrement.

Types of pointer in detail



Pointer to an Array or Array Pointer

Pointer to an array or array pointer

```
#include<stdio.h>

int main()
{
  int arr[5] = { 1, 2, 3, 4, 5 };
  int *ptr = arr;

  printf("%p\n", ptr);
  return 0;
}
```

$$\underbrace{*\ Ptr}_{\downarrow}$$

1

- *ptr* that points to the 0th element of the array.
- We can also declare a pointer that can point to whole array.

Output:
0x7ffe98c0faf0

Pointer to an array or array pointer

Syntax:

```
data_type (*var_name)[size_of_array];
```

Example

*int (*ptr)[10];*

ptr is pointer that can point to an array of 10 integers

data_type is the type of data that the array holds.

var_name is the name of the pointer variable.

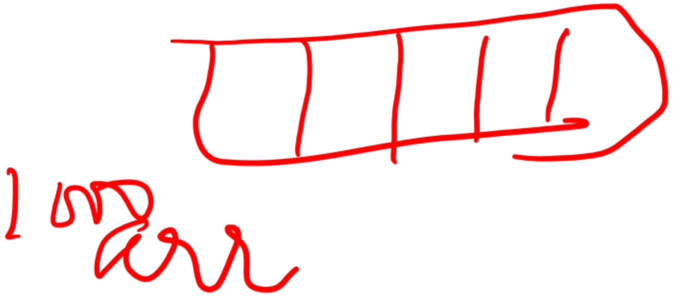
size_of_array is the size of the array to which the pointer will point.

Difference between pointer to an integer and pointer to an array of integers.

```
#include<stdio.h>
int main()
{
int *p; // Pointer to an integer
int (*ptr)[5]; // Pointer to an array of 5 integers
int arr[5];
p = arr; // Points to 0th element of the arr.
    ptr = &arr; // Points to the whole array arr.
    printf("p = %p, ptr = %p\n", p, ptr);
    p++;
    ptr++;
    printf("p = %p, ptr = %p\n", p, ptr);
    return 0;
}
```

1000 1000

1004 1020



Output:

```
p = 0x7ffd199ce0b0, ptr = 0x7ffd199ce0b0
p = 0x7ffd199ce0b4, ptr = 0x7ffd199ce0c4
```

Sizes of pointer of array

```
#include<stdio.h>
int main()
{
    int arr[] = { 3, 5, 6, 7, 9 };
    int *p = arr;
    int (*ptr)[5] = &arr;
    printf("p = %p, ptr = %p\n", p, ptr);
    printf("*p = %d, *ptr = %p\n", *p, *ptr);
    printf("sizeof(p) = %lu, sizeof(*p) = %lu\n", sizeof(p), sizeof(*p));
    printf("sizeof(ptr) = %lu, sizeof(*ptr) = %lu\n", sizeof(ptr), sizeof(*ptr));
    return 0;
}
```

Output:

```
p = 0x7fff5dd31d40, ptr = 0x7fff5dd31d40
*p = 3, *ptr = 0x7fff5dd31d40
sizeof(p) = 8, sizeof(*p) = 4
sizeof(ptr) = 8, sizeof(*ptr) = 20
```

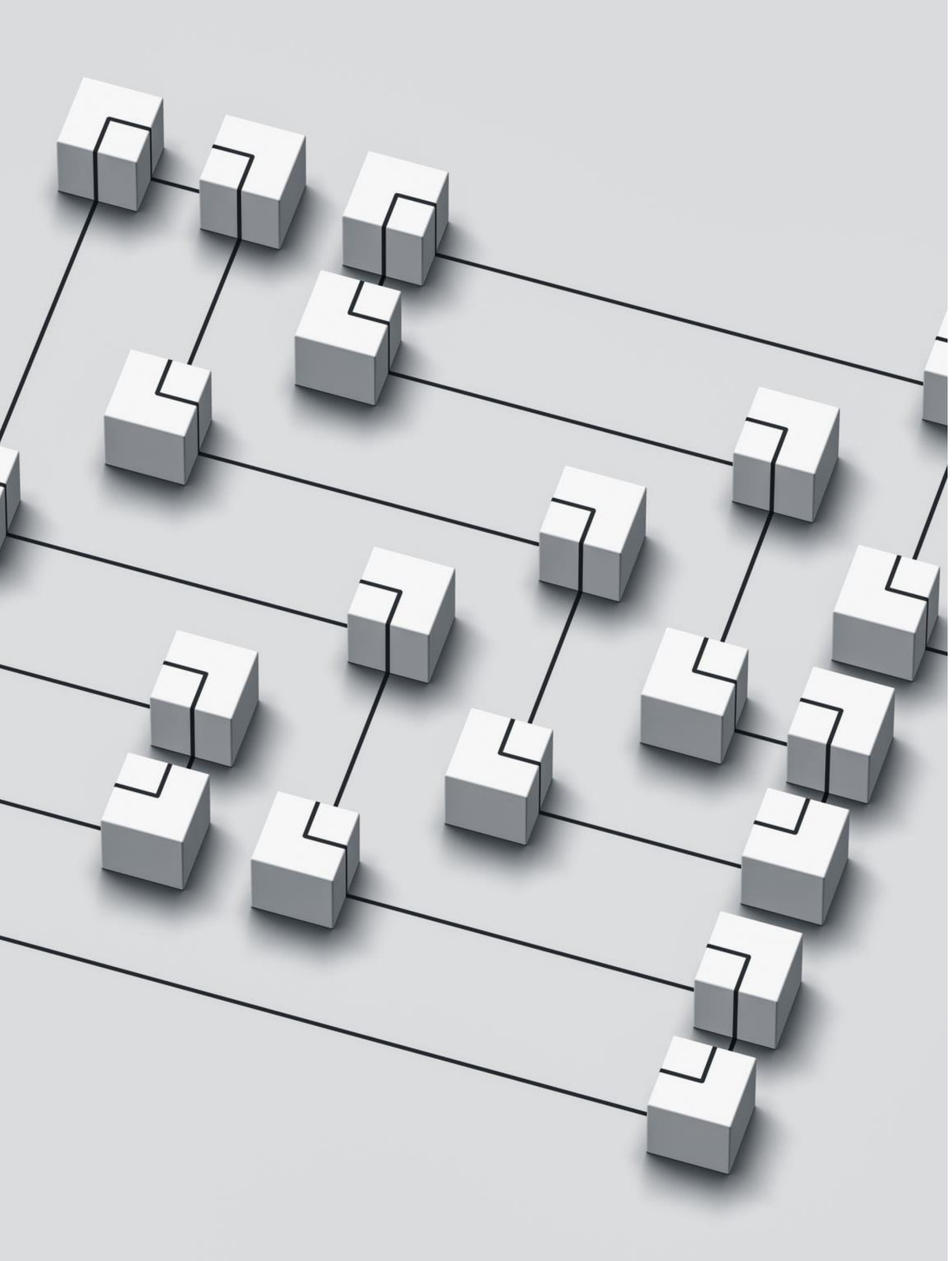
Announcement: Quiz 2

- Quiz 2 is on **24th Oct (Thursday)- from 12:30pm to 1pm**.

- Syllabus will be including Pointers that I have covered till 17th Oct. (**Conditional statements, Loops, arrays, functions, Macro & Inline function, recursion, pointers**)

- **5 to 6 questions**: MCQs, Short answer question & coding question

- **10 to 15 marks**



Upcoming Slides

- Function Pointers
- Dynamic Memory allocation