Slot 04 - Dynamic Structures

Presenter:

Dr. LE Thanh Tung

Content

1. Pointer with File
2. Pointer in Struct
3. Pointer to Pointer

Pointer

- Pointer holds memory address of a variable
- Reference operator: & -> address of variable
- Dereference operator: * -> value of corresponding address

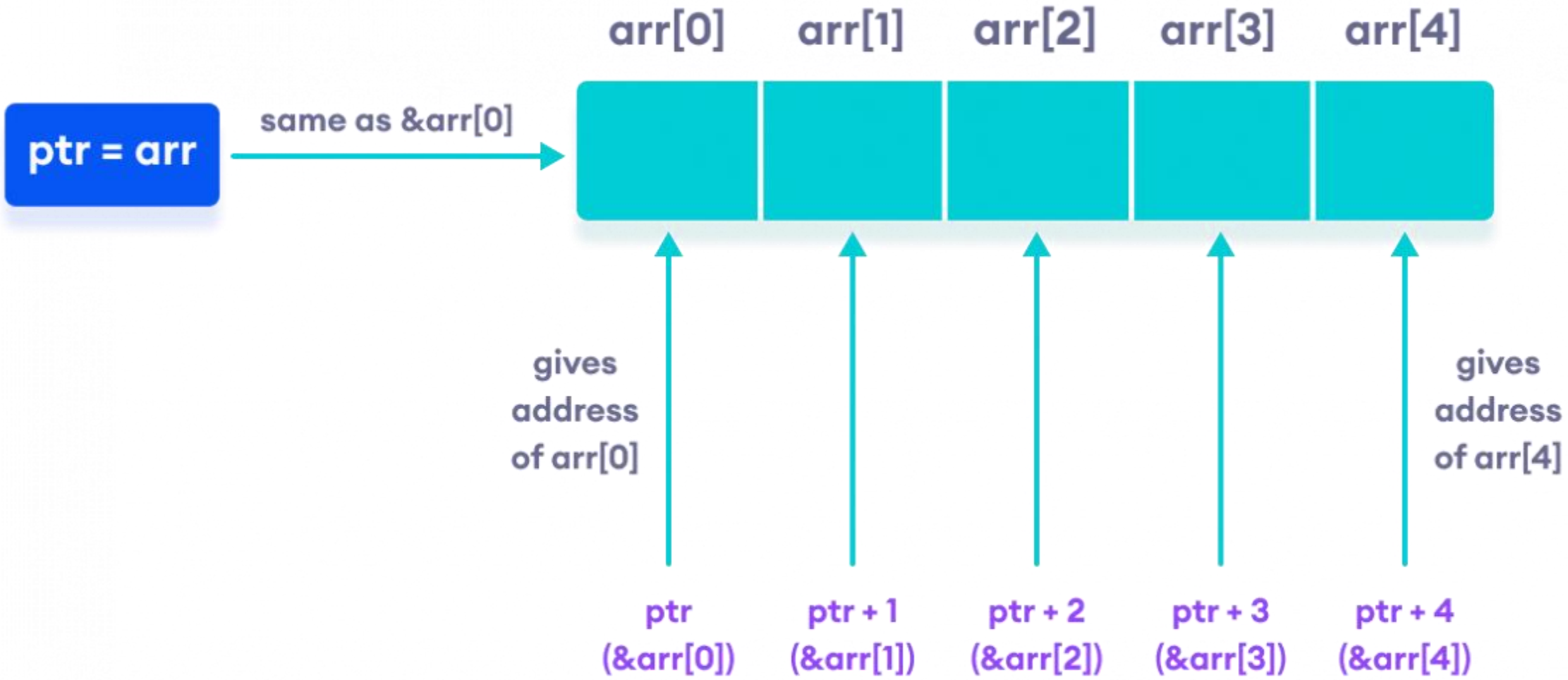
```
int* pointVar, var;
var = 5;

// assign address of var to pointVar
pointVar = &var;

// access value pointed by pointVar
cout << *pointVar << endl;   // Output: 5
```

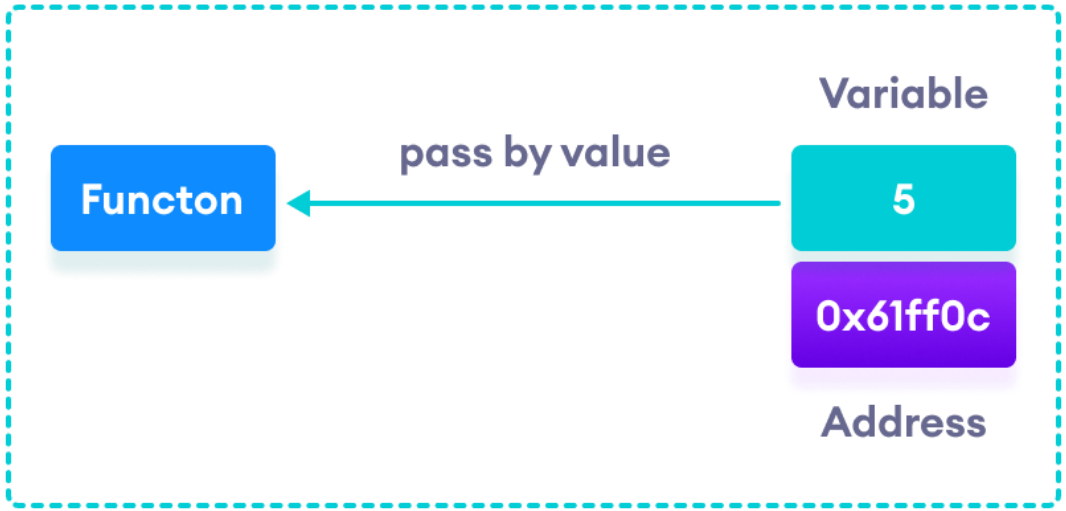
Common mistakes

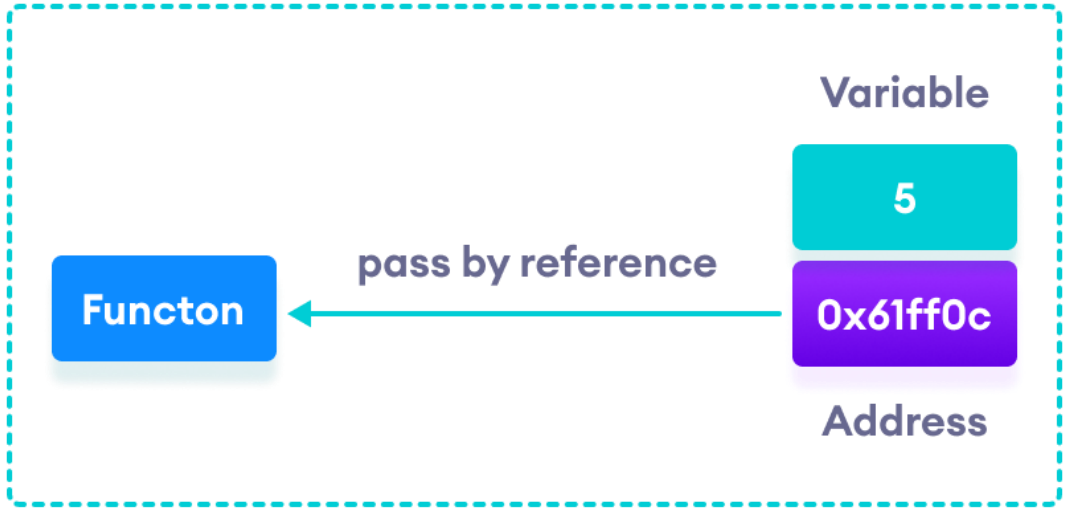
```
int var, *varPoint;

// Wrong!
// varPoint is an address but var is not
varPoint = var;

// Wrong!
// &var is an address
// *varPoint is the value stored in &var
*varPoint = &var;
```

Common mistakes

```
int var, *varPoint;

 // Correct!
 // varPoint is an address and so is &var
 varPoint = &var;

  // Correct!
 // both *varPoint and var are values
 *varPoint = var;
```


arr[0]
arr[1]
arr[2]
arr[3]
arr[4]
ptr = arr
same as &arr[0]
gives address of arr[0]
gives address of arr[4]
ptr (&arr[0])
ptr + 1 (&arr[1])
ptr + 2 (&arr[2])
ptr + 3 (&arr[3])
ptr + 4 (&arr[4])


Variable
pass by value
Functon
5
0x61ff0c
Address


Variable
5
pass by reference
Functon
0x61ff0c
Address

Pointers and Functions

```
// function definition to swap numbers
void swap(int* n1, int* n2) {
    int temp;
    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
```

`*n1` and `*n2` gives the value stored at address `n1` and `n2` respectively.

Since `n1` and `n2` contain the addresses of `a` and `b`, anything is done to `*n1` and `*n2` will change the actual values of `a` and `b`.

```
int main()
{

    // initialize variables
    int a = 1, b = 2;

    cout << "Before swapping" << endl;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;

    // call function by passing variable addresses
    swap(&a, &b);

    cout << "\nAfter swapping" << endl;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    return 0;
}
```

Pointers and Allocation

```
// declare an int pointer
int* pointVar;

// dynamically allocate memory
// for an int variable
pointVar = new int;

// assign value to the variable memory
*pointVar = 45;

// print the value stored in memory
cout << *pointVar; // Output: 45

// deallocate the memory
delete pointVar;
```

```
int num;
cout << "Enter total number of students: ";
cin >> num;
float* ptr;

// memory allocation of num number of floats
ptr = new float[num];

cout << "Enter GPA of students." << endl;
for (int i = 0; i < num; ++i) {
  cout << "Student" << i + 1 << ": ";
  cin >> *(ptr + i);
}
```

```
cout << "\nDisplaying GPA of students." << endl;
for (int i = 0; i < num; ++i) {
  cout << "Student" << i + 1 << ": " << *(ptr + i) << endl;
}

// ptr memory is released
delete[] ptr;

return 0;
```

Pointers in Structure

- ptr is pointing to Distance d -- *ptr is to deference the value in d

```
int main() {
    Distance *ptr, d;

    ptr = &d;

    cout << "Enter feet: ";
    cin >> (*ptr).feet;
    cout << "Enter inch: ";
    cin >> (*ptr).inch;

    cout << "Displaying information." << endl;
    cout << "Distance = " << (*ptr).feet << " feet " << (*ptr).inch << " inches";

    return 0;
}
```

```
#include <iostream>
using namespace std;

struct Distance {
    int feet;
    float inch;
};
```

- (*ptr).inch and d.inch are equivalent
- How about *ptr.inch ? WRONG
- Both the member access operator (.) has a higher precedence than the dereference (*)
- if we are using pointers, it is far more preferable to access struct members using the -> operator

```
cout << "Enter feet: ";
cin >> (*ptr).feet;
cout << "Enter inch: ";
cin >> (*ptr).inch;
```

```
ptr->feet is same as (*ptr).feet
ptr->inch is same as (*ptr).inc
```

Pointers and Allocation

- Now, to allocate an array of structures dynamically
- In this case, how would we access the first Distance's feet
  - ptr[0].feet
  - ***Notice that the -> operator would be incorrect in this case because ptr[0] is not a pointer variable***

```
#include <iostream>
using namespace std;

struct Distance {
    int feet;
    float inch;
};

int main() {
  Distance *ptr;
  int n = 2;
  ptr = new Distance[5];

  for (int i = 0; i < n; i++){
    cout << "Enter feet: ";
    cin >> ptr[i].feet;
    cout << "Enter inch: ";
    cin >> ptr[i].inch;
  }
  cout << "Displaying information:" << endl;
  for (int i = 0; i < n; i++)
    cout << "Distance = "
         << ptr[i].feet << " feet "
            << ptr[i].inch << " inches" << endl;

  delete[] ptr;
  return 0;
}
```

Pointers and Allocation

- What this tells us is that the -> operator expects a pointer variable as the first operand

```
#include <iostream>
using namespace std;

struct Distance {
    int feet;
    float inch;
};

int main() {
  Distance *ptr;
  int n = 2;
  ptr = new Distance[5];

  for (int i = 0; i < n; i++){
    cout << "Enter feet: ";
    cin >> ptr[i].feet;
    cout << "Enter inch: ";
    cin >> ptr[i].inch;
  }
  cout << "Displaying information:" << endl;
  for (int i = 0; i < n; i++)
    cout << "Distance = "
         << ptr[i].feet << " feet "
            << ptr[i].inch << " inches" << endl;

  delete[] ptr;
  return 0;
}
```

Pointers in Struct

- We can include dynamically allocated array in struct to avoid wasting the memory
- However, remember to delete both struct object and its members
- Q: Why the length of title is increased by 1?

```
struct Video {
    char* title;
    char category[5];
    int quantity;
};
```

```
void setTitle(Video*& v){
    char tmp[100];
    cin.get(tmp, 100);
    v = new Video;
    v->title = new char[strlen(tmp) + 1];
    strcpy(v->title, tmp);
}

int main() {
    Video* ptr;
    setTitle(ptr);
    cout << "The title of video: " << ptr->title;
    delete[] ptr->title;
    delete ptr;

    return 0;
}
```

- An 2d-array is also stored in sequential form for memory

2d-Array & Pointer

- Therefore, we can use the pointer to hold the address of 2d array in C++

```
int main() {
    int a[2][3];
    int n = 2, m = 3;
    int* ptr = (int*) a;
    for (int i = 0; i < n*m; i++)
      cin >> *(ptr + i);

    for (int i = 0; i < n; i++){
      for (int j = 0; j < m; j++)
        cout << a[i][j] << "\t";
      cout << endl;
  }
```

2d-Array with Pointer

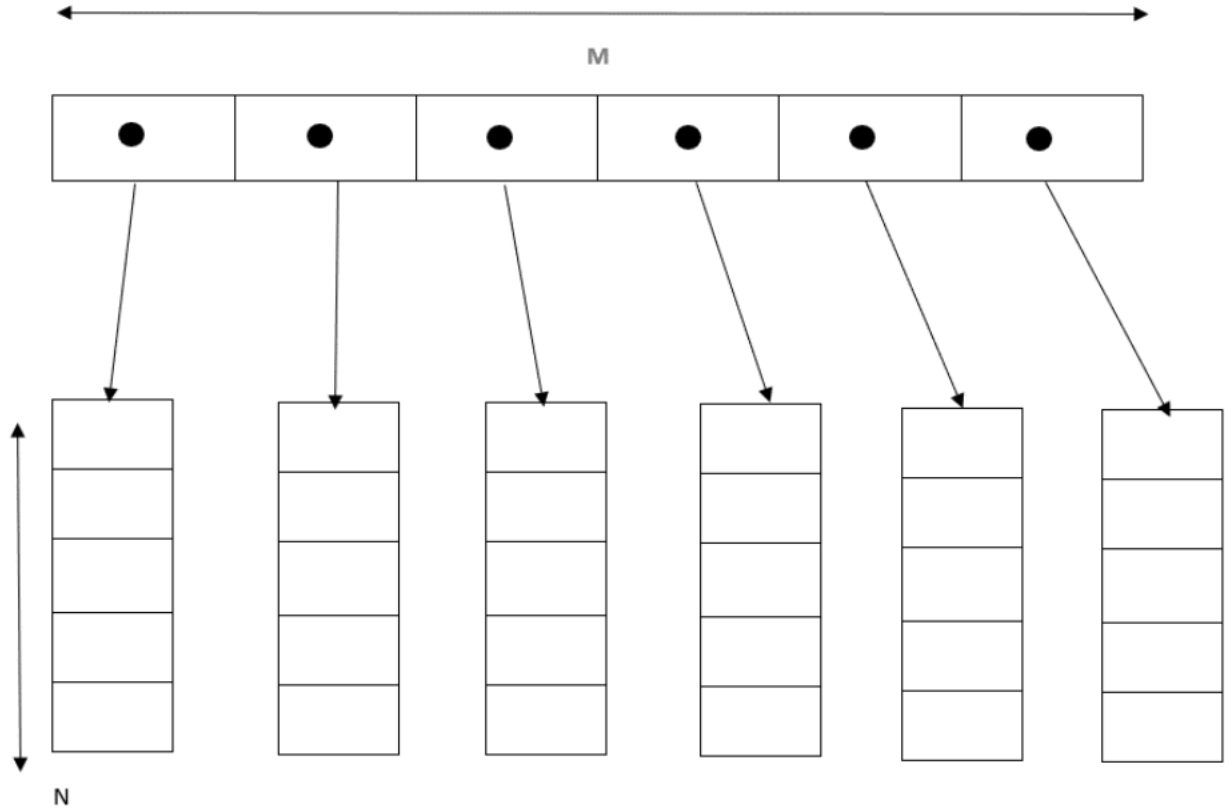
- Dynamically allocated 2D-array: convert the 2d Array into 1d array and using pointer

```
int* a = new int[M * N];//dynamically allocating memory
```

```
//displaying the 2D array
for (int i = 0; i < M; i++)
{
    for (int j = 0; j < N; j++) {
        cout << *(a + i*N + j) << " ";
    }
    cout << endl;
}
```

2d-Array with Pointer

- Option 2: Using an array of pointers to create a 2D array dynamically
- In this approach, we can dynamically create an array of pointers of size M and dynamically allocate memory of *size N** for each row of the 2D array



2d-Array with Pointer

- Option 2: Using an array of pointers to create a 2D array dynamically
- In this approach, we can dynamically create an array of pointers of size M and dynamically allocate memory of *size N** for each row of the 2D array

```
// creating an array of pointers
// of size M dynamically using pointers
int** a = new int*[M];

// dynamically allocating memory of size `N`
//for each row
for (int i = 0; i < M; i++) {
    a[i] = new int[N];
}
```

2d-Array with Pointer

```
// creating an array of pointers
// of size M dynamically using pointers
int** a = new int*[M];

// dynamically allocating memory of size `N`
//for each row
for (int i = 0; i < M; i++) {
    a[i] = new int[N];
}
```

Remember to deallocate each row first, and the matrix later

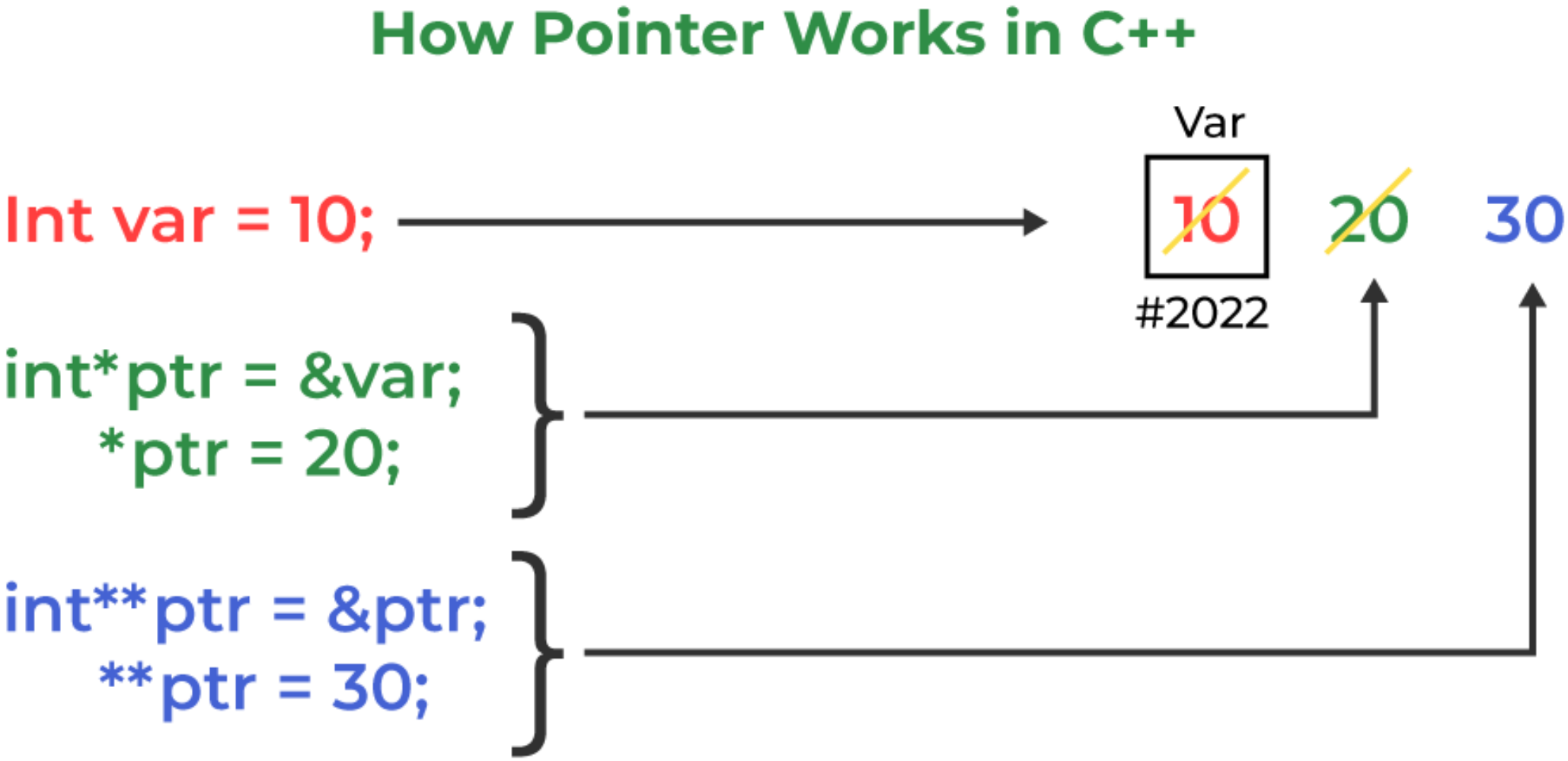
```
// deallocate memory
// using the delete operator
for (int i = 0; i < M; i++) {
    delete[] a[i];
}
delete[] a;
```

Pointer to Pointer – Double Pointer

- A pointer to a pointer is a form of multiple indirection or a chain of pointers

- Declaration:

```
int **var;
```


How Pointer Works in C++
Var
Int var = 10;
10
20
30
#2022
int*ptr = &var;
*ptr = 20;
int**ptr = &ptr;
**ptr = 30;

Pointer to Pointer – Double Pointer

```
int main () {
   int  var;
   int  *ptr;
   int  **pptr;
   var = 3000;
   // take the address of var
   ptr = &var;
   // take the address of ptr using address of operator &
   pptr = &ptr;
   // take the value using pptr
   cout << "Value of var :" << var << endl;
   cout << "Value available at *ptr :" << *ptr << endl;
   cout << "Value available at **pptr :" << **pptr << endl;
   return 0;
}
```

- Given 2D array A, count all prime number in this array.

- Given 2 1D arrays a and b. Generate the matrix c that c[i][j] = a[i] + b[j]

```
int** generateMatrix2(int* a, int* b, int na, int nb)
```

THANK YOU
for YOUR ATTENTION