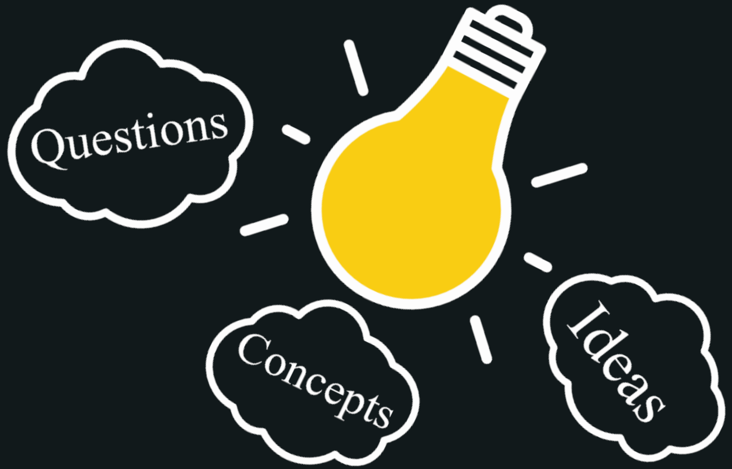


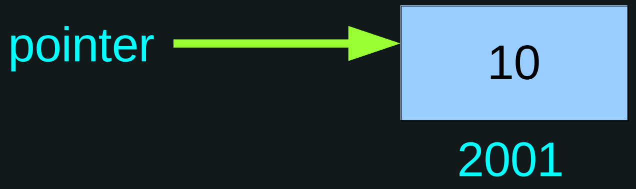
Pointers

C++ Programming

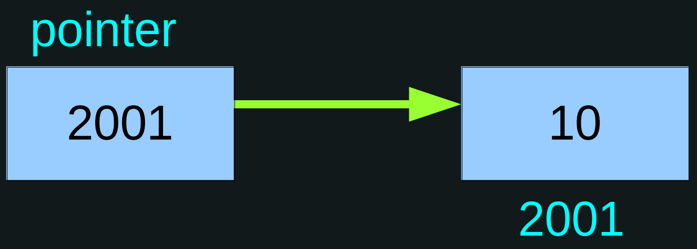
VEDINESH

What is Pointer ?

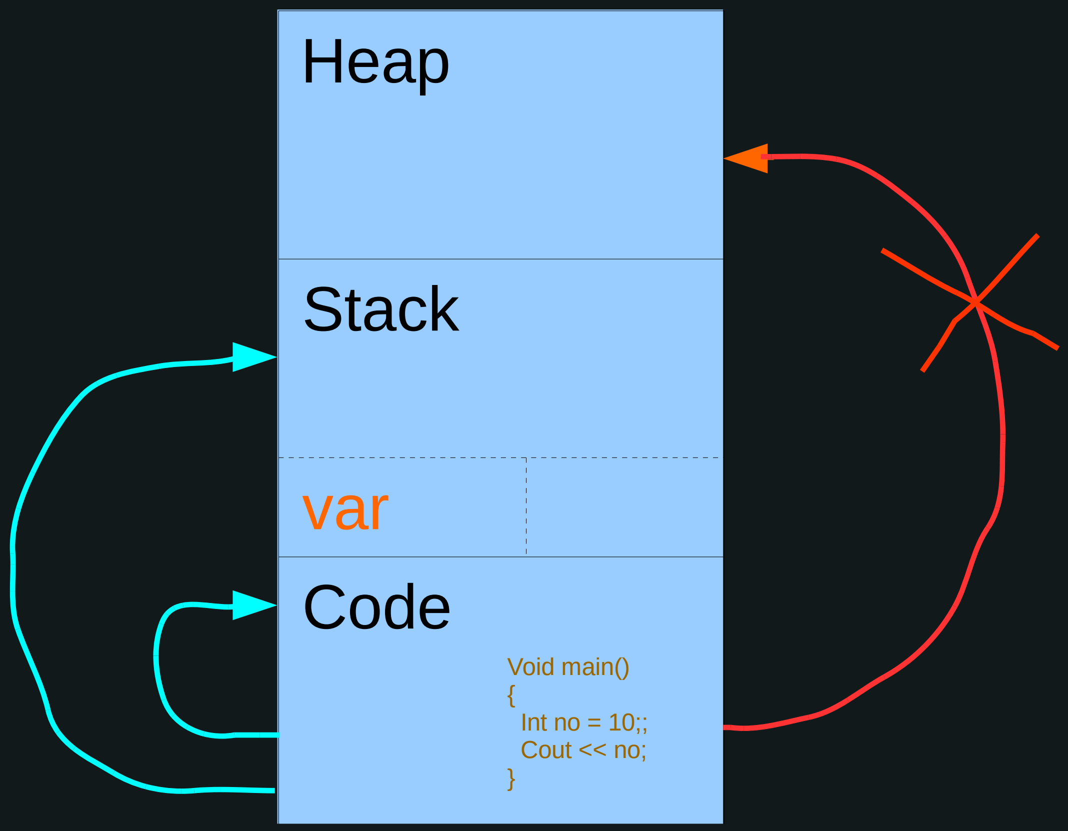
Pointer points to a memory



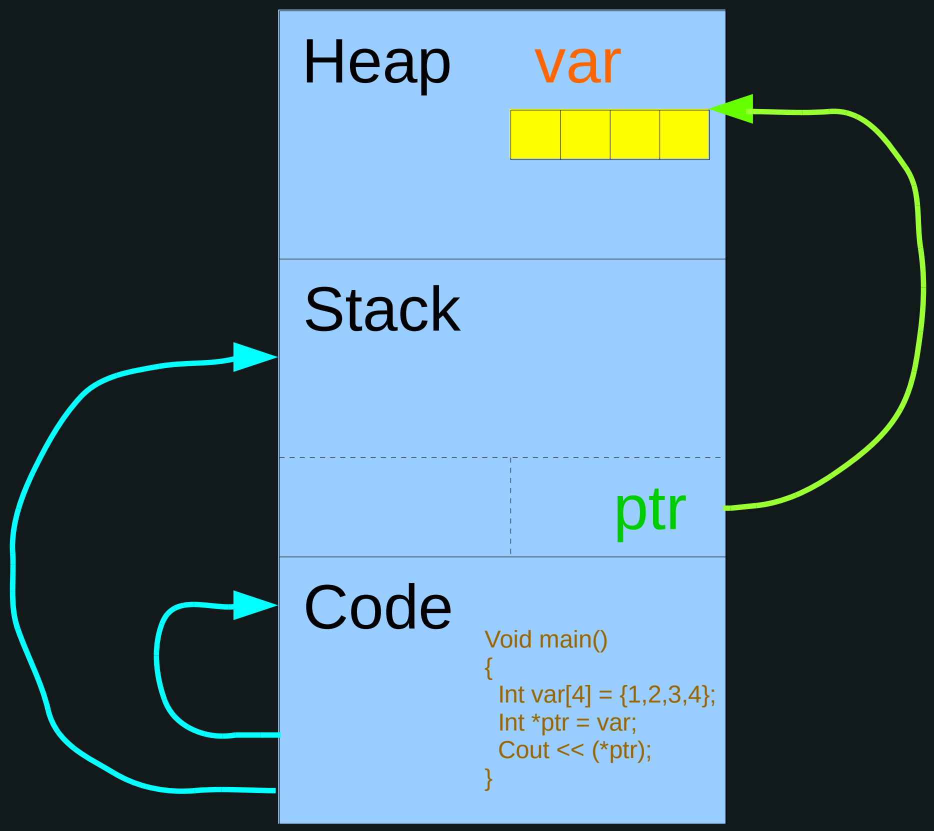
Pointer knows how to reach to that memory location.



Why Pointers ?



```
Void main()
{
  Int no = 10;;
  Cout << no;
}
```



```
Void main()
{
  Int var[4] = {1,2,3,4};
  Int *ptr = var;
  Cout << (*ptr);
}
```

Why Pointers ?

Pointer Uses

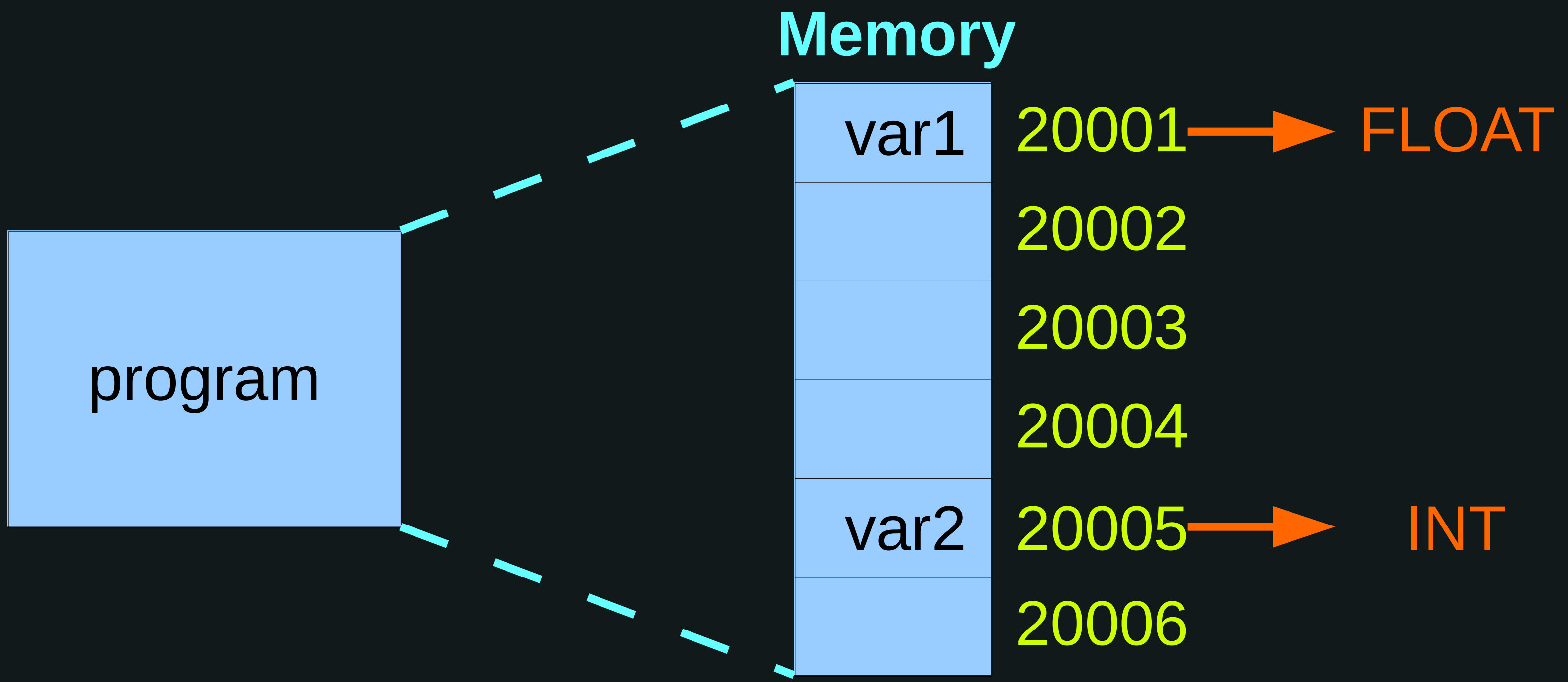
1. Directly Accessing Memory
2. Accessing Array Elements
3. Passing Arrays and strings to function
4. For creating Data Structures like linked list

Use Properly

Pointer Usage

virtual functions --- new Operator ----- this pointer

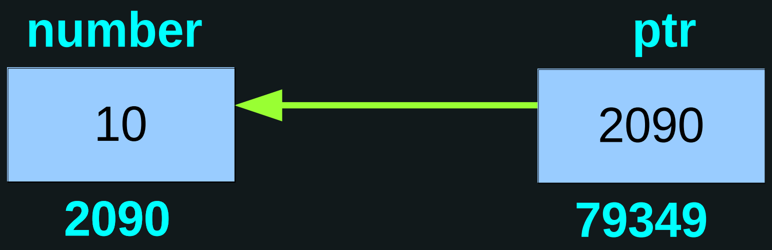
Program in Memory



Pointers Notation

```
Data_Type *POINTER_NAME;
```

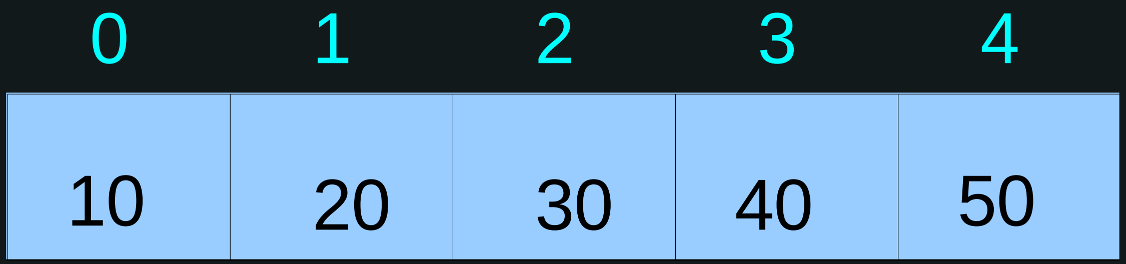
```
int  *ptr = NULL;
```



```
void main ( )
{
   int number = 10;
   int *ptr = NULL;
   ptr = &number;

   cout << ptr;                //pointer address

   cout << ( *ptr );           // value pointed
                                  by pointer
}
```

Pointer And Arrays

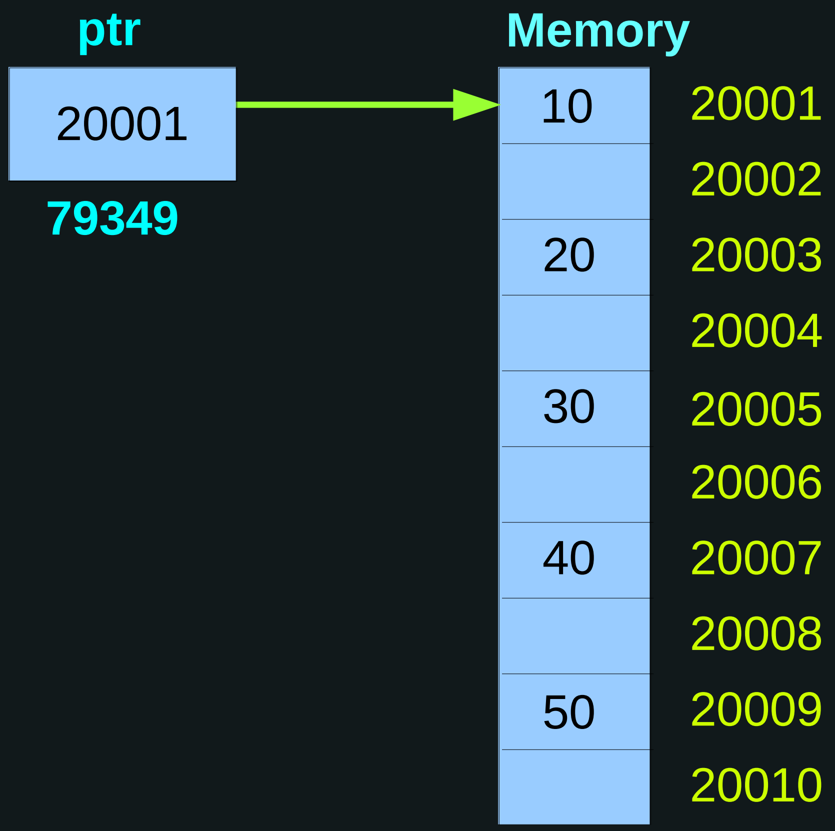
```
void main ( )
{
   int arr[5]=  { 10, 20, 30, 40, 50 };

   for ( int i =0; i<5; i++ )
   {
      cout << arr[i] << endl;
   }
}
```



Pointer And Arrays

```
void main ( )
{
   int arr[5]=  { 10, 20, 30, 40, 50 };
   int *ptr = arr;

   for ( int i =0; i<5; i++ )
   {
      cout << *( ptr + i ) << endl;
   }
}
```

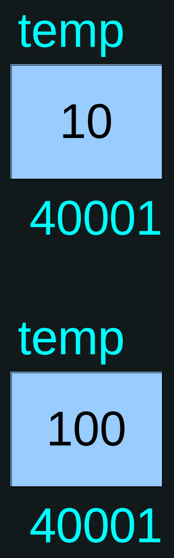
ptr (20001)

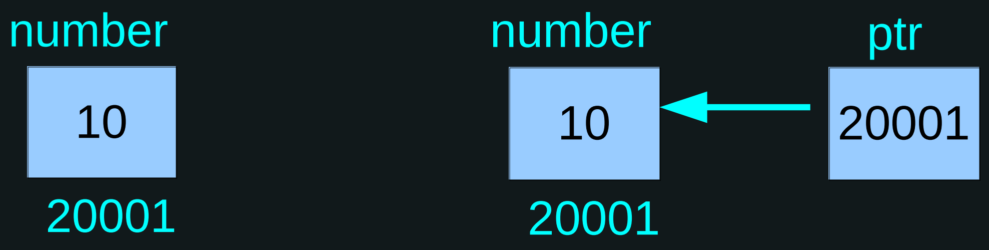


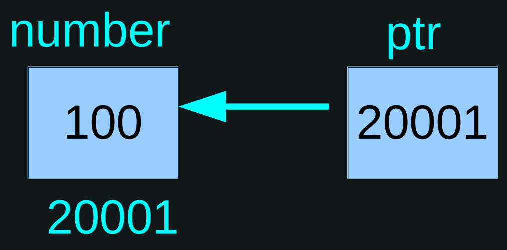
Pointers And Function

```
void main( )
{
  int number = 10;

  cout << number;

  square ( &number );

  cout << number;
}
```

```
void square( int *ptr )
{
   int temp = *ptr;

   temp = temp*temp;

  *ptr = temp;
}
```





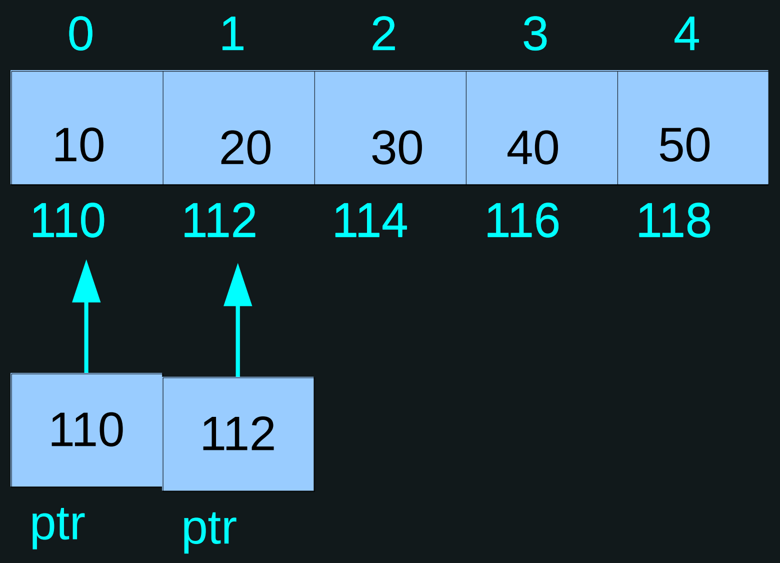


Pointers And Function (Passing Array)

VEDINESH

```
#include<iostream>
using namespace std;
const int MAX = 5;

void main( )
{
  int number[MAX] =  {10,20,30,40,50};

  printArray ( number );   // &number[0]

}
```

```
void printArray( int *ptr )
{
   for( int i=0; i<MAX; i++ )
   {
            cout << *ptr++;
    }
}
```



Memory Management

Static Memory Allocation

Static Memory Allocation, is used when we know amount of memory needed (like we have upper limit.

Static Memory Allocation, is done automatically by your compiler.

Static Memory Allocation, is allocated on STACK.

Dynamic Memory Allocation

Dynamic Memory Allocation, is used when we don't know amount of memory needed (like no upper limit.

Dynamic Memory Allocation, is done manually by programmer.

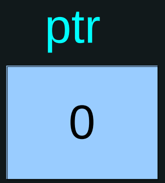
Dynamic Memory Allocation, is allocated on HEAP.

new - allocation / delete - deallocation

Memory Management: new

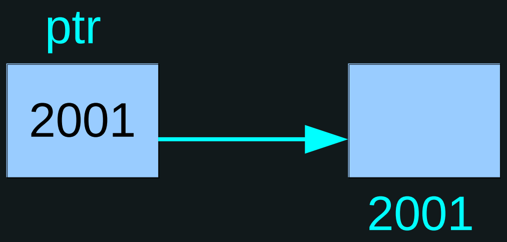
> Syntax to use new operator:

```
int  *ptr  =  NULL;
```



pointer is initialized with NULL

```
ptr  =  new int;
```



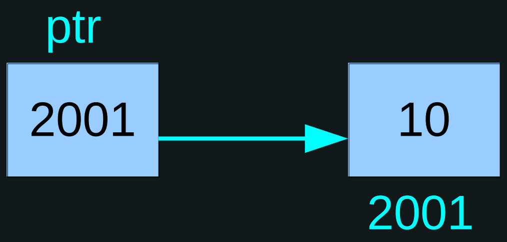
Request for memory

or

```
int *ptr  =  new int;
```

> Syntax to Initialize memory:

```
int *ptr  =  new int(10);
```



Initialize memory using new

Memory Management: new

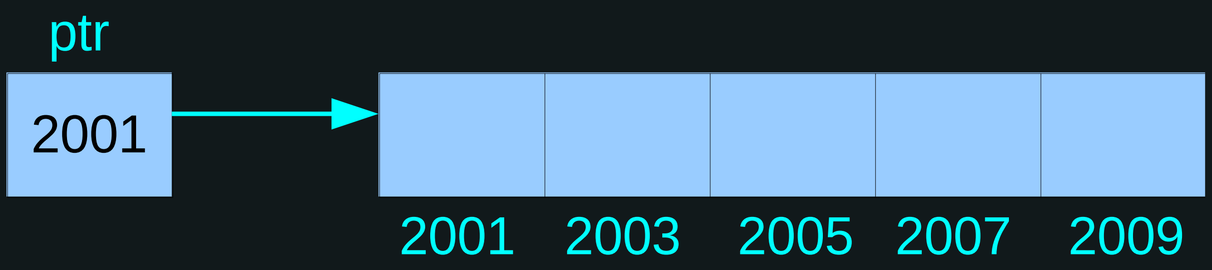
> Syntax to Allocate block of memory:

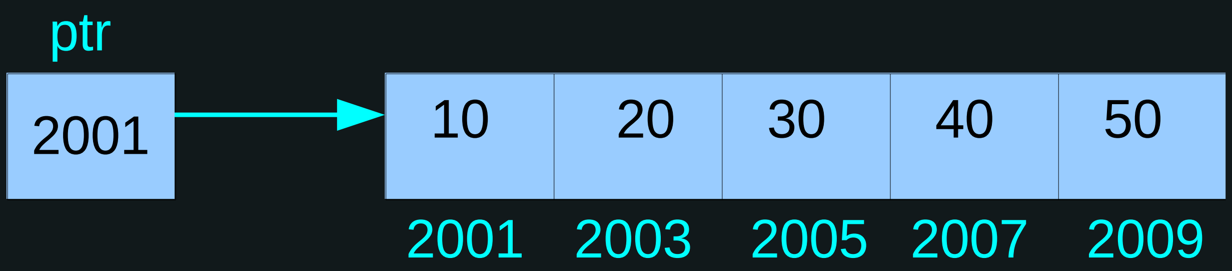
```
int *ptr  =  new int [5];
```

```
*ptr  =  10;
```

```
*( ptr + 1 )  =  20;
```

```
*( ptr + 2 )  =  30;
```





Memory Management: new

```
class Test
{
private:
   int data;
public:
   void setData( int set )
   { data = set; }

   int getValue( )
   { return data; }
 };
```

```
void main( )
{
   Test *t2;              // pointer to Test
   t2 = new Test;       //  points to new Test Object

  t2->setData(10);

  cout << t2->getData( );

}
```

Memory Management: delete

Delete is use to deallocate dynamically allocated memory.

> Syntax to Deallocate memory pointed by pointer:

```
int *ptr = new int;          →          delete ptr;
```

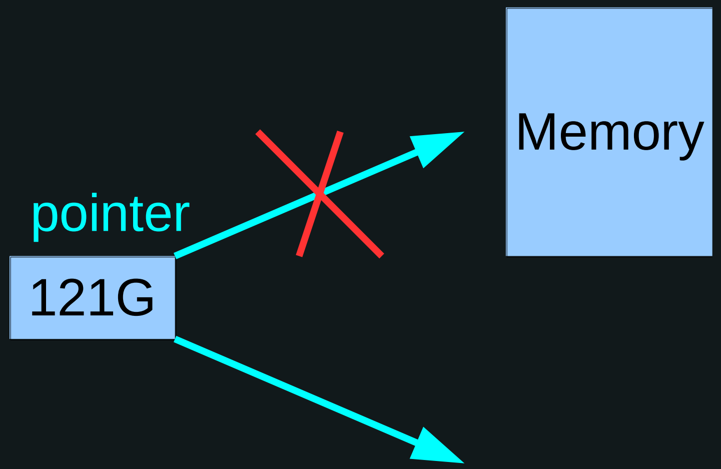
> Syntax to Deallocate block of memory pointed by pointer:

```
int *ptr = new int[10];      →          delete[] ptr;
```

Pointers Limitations

VEDINESH

1. Uninitialized Pointer.



problem

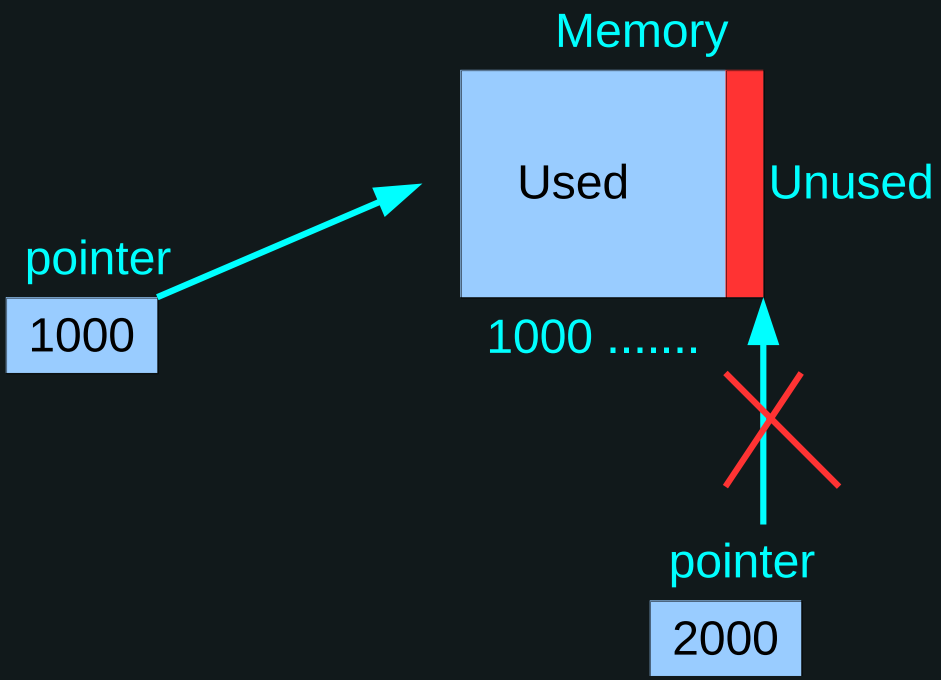
```
dataType *pointerName;
```

solution

```
dataType *pointerName = &variable;
dataType *pointerName = NULL;
dataType *pointerName = new dataType;
```

Pointers Limitations

2. Memory Leaks.



problem (not using)

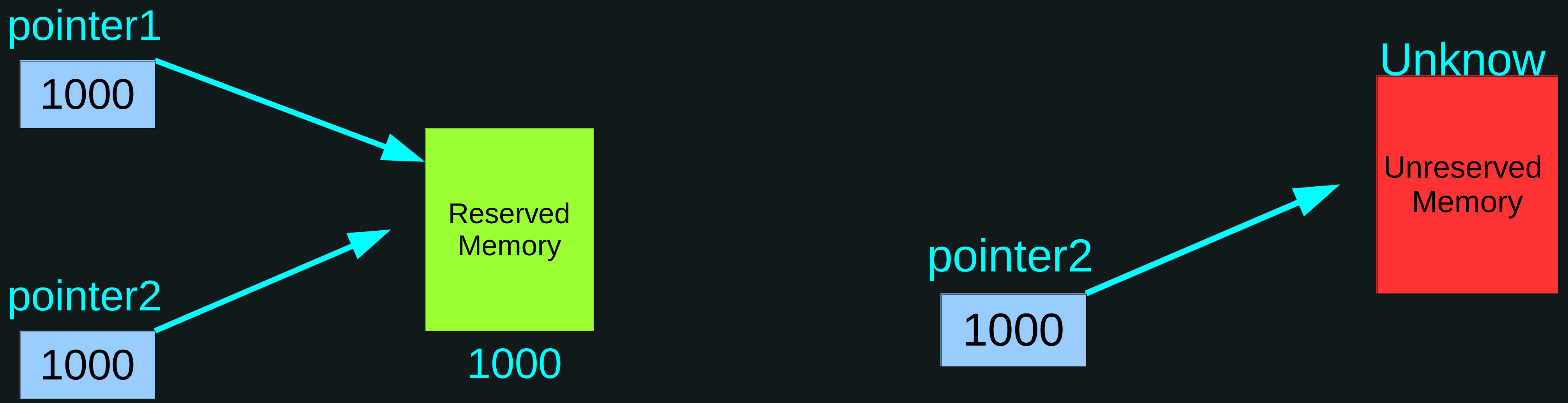
```
delete pointerName;
```

solution (use)

```
delete pointerName;
delete [] pointerName;
```

Pointers Limitations

3. Dangling Pointer.



Program Crash