- ## Dynamic Memory (C++)
  - Memory is divided into 2 parts
    - **STACK**: variables declared inside function will take up memory
    - **HEAP**: unused memory can be used to allocate memory dynamically
  - **new** operator: used to allocate memory ar run time within heap for variable of a given type, which returns the address of the space allocated
  - **delete** operator: deallocates memory allocated by new operator
  - following syntax to use **new** operator to allocate memory dynamically

```
new data-type;
```

    - ➔ data-typee-here could be built in data type including array or user defined data types and class or structure
    - can define a ptr to type double and then request that the memory be allocated at execution time

```
double* pvalue  = NULL; // Pointer initialized with null
pvalue  = new double;   // Request memory for the variable
```

    - memory may not have been allocated successfully if free store had been used up
    - good practice to check if **new** operator Is returning NULL ptr and take appropriate action in code below

```
double* pvalue  = NULL;
if( !(pvalue  = new double )) {
        cout << "Error: out of memory." <<endl;
        exit(1);
}
```

  - **malloc():** advantage of **new** is that it allocates memory and contructs objects which is prime purpose of C++
  - **delete** operator: used to free up memory allocated

```
        delete pvalue;      // Release memory pointed to by pvalue
```

  - example of operators

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

int main () {
  double* pvalue  = NULL; // Pointer initialized with null
  pvalue  = new double;   // Request memory for the variable

  *pvalue = 29494.99;     // Store value at allocated address
  cout << "Value of pvalue : " << *pvalue << endl;

  delete pvalue;        // free up the memory.

  return 0;
}
```

    - Value of pvalue : 29495

- Dynamic Memory Allocation for Arrays
  - ➢ Allocate memory for array of chars

```
char* pvalue  = NULL;       // Pointer initialized with null
pvalue  = new char[20];      // Request memory for the variable
```

  - ➢ To remove array

```
delete [] pvalue;          // Delete array pointed to by pvalue
```

  - ➢ Allocation for multi-dimensional array

```
double** pvalue  = NULL;     // Pointer initialized with null
pvalue  = new double [3][4];  // Allocate memory for a 3x4 array
```

  - ➢ Release memory for multi-dimensional array

```
delete [] pvalue;          // Delete array pointed to by pvalue
```

- Objects are no different from simple data types

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

class Box {
  public:
    Box() {
      cout << "Constructor called!" <<endl;
    }
    ~Box() {
      cout << "Destructor called!" <<endl;
    }
};
int main() {
  Box* myBoxArray = new Box[4];
  delete [] myBoxArray; // Delete array

  return 0;
}
```

- Compile shows as follows

```
Constructor called!
Constructor called!
Constructor called!
Constructor called!
Destructor called!
Destructor called!
Destructor called!
Destructor called!
```