Lecture 6

EXCEPTION HANDLING

&

DYNAMIC SAFE ARRAYS

September 16, 2021

Thursday

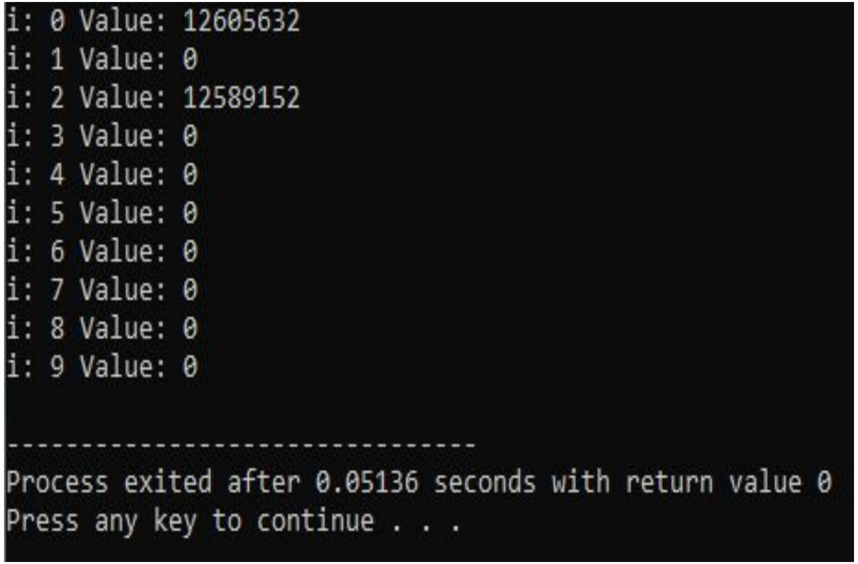
NULL IN C/C++

1. NULL Character
   a. ‘\0’ at the end of character array to let compiler know, string has ended.
2. NULL Pointer
   a. ‘0’ to let compiler know that pointer is not pointing at any memory address.
3. NULL Statement
   a. ‘;’ just a statement with missing expression, it does nothing.
4. Can’t use none in C/C++.
5. There is no default initialization for primitive data types in C/C++.

JAVA & PYTHON

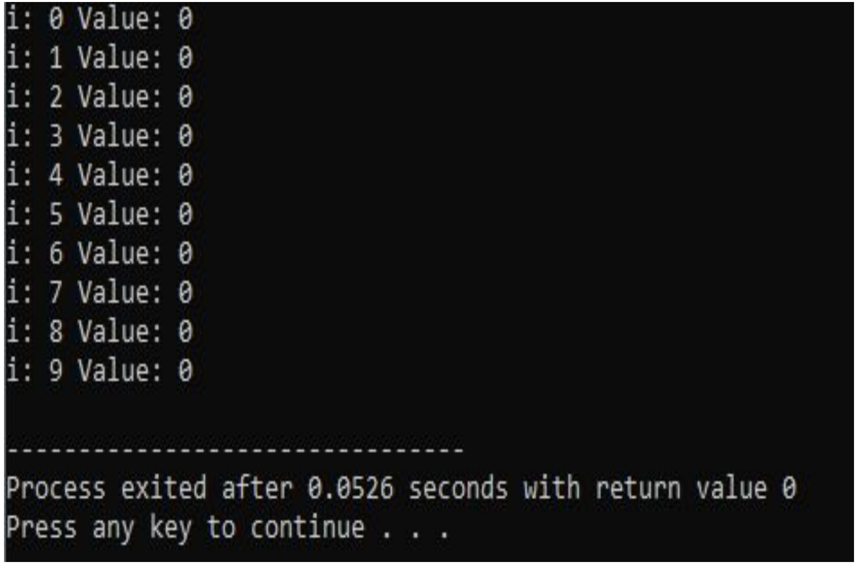
- Java
  - Literal (true/false).
  - Small caps: **null**.
  - Can only be used with reference variables.
  - Compile time error if used with primitive data variables.
- Python
  - There is no NULL/null in python.
  - There is **None** in python.
  - Can't use none.
  - Use to check if an object/variable is initialized.

INITIALIZATION



```
i: 0 Value: 12605632
i: 1 Value: 0
i: 2 Value: 12589152
i: 3 Value: 0
i: 4 Value: 0
i: 5 Value: 0
i: 6 Value: 0
i: 7 Value: 0
i: 8 Value: 0
i: 9 Value: 0


--------------------------------
Process exited after 0.05136 seconds with return value 0
Press any key to continue . . .
```



```
i: 0 Value: 0
i: 1 Value: 0
i: 2 Value: 0
i: 3 Value: 0
i: 4 Value: 0
i: 5 Value: 0
i: 6 Value: 0
i: 7 Value: 0
i: 8 Value: 0
i: 9 Value: 0


--------------------------------
Process exited after 0.0526 seconds with return value 0
Press any key to continue . . .
```

EXCEPTION HANDLING

Exception is a problem that occurs infrequently at runtime.

- Exception handling is to develop a **fault-tolerant programs**, that can resolve exceptions.
- If the problem is not severe
  - Don’t let your program crash, user doesn’t know if an error has occurred.
- If the problem is severe
  - Don’t let program continue
  - Notify the user of the problem
  - Then terminate *gracefully*.

EXCEPTION HANDLING | WHY & WHEN

When

From **the inception of the project**

Why

while working on a **project with a large team**.

Saves time & Effort.

Later it become difficult and costly.

EXCEPTION HANDLING | ELEMENTS

- **try block**
  - Contains the code that might throw an exception.
- **throw**
  - When an error occurs in a program throw an exception with a message.
- **catch block**
  - Contains the code that handles the exception if one occurs.
  - We can use multiple catch blocks for different types of exceptions.
  - Recieves the error message thrown by the throw or the exception.

EXCEPTION HANDLING | ELEMENTS

```
int main ( ) {
    try {
        ....
        if (condition is true)
            throw exception;
    } catch (type exception ) {
        ...
        //do something about the exception.
        ...
    }
    return 0;
}
```

EXCEPTION HANDLING

- If no exception is thrown
  - the try block executes completely
  - catch block is ignored.
- If an exception occurs
  - try block terminates immediately
  - catch block is executed.
  - all the variables of try block are now out of scope.

CUSTOM EXCEPTION CLASS

- C++ provides standard base class **exception** for exceptions in the C++ Standard Library. Defined in **<exception>** header.
- C++ also provides Standard Library class **runtime_error** for representing the run-time errors. Defined in **<stdexcept>** header.
- For custom exception class
  - inherit **runtime_error** class
  - define only a **constructor**.
- Every exception class which inherits from **exception** contains the virtual function ***what***
  - returns an exception object's **error message**.

CUSTOM EXCEPTION CLASS

```
#include<stdexcept>          // contains runtime_error
using namespace std;


class CustomException : public runtime_error {
    public:
        CustomException ( ) : runtime_error (“Something bad has happened”)
          {

          }
}
```

REDEFINING OUR INDEXING

```
class  DynamicArray {
...........
public:
        int& operator [] (int index) {
        int *pnewa;
        if (index >= length) {
        pnewa = new int[index + 10];

        for (int i = 0; i < nextIndex; i++)
                pnewa[i] = pa[i];

        for (int j = nextIndex; j < index + 10; j++)
                pnewa[j] = 0;
};
```

```
                  length = index + 10;
                  delete [] pa;
                  pa = pnewa;
                  }

                  if (index > nextIndex)
                  nextIndex = index + 1;
                  return *(pa + index);
          }

};
```

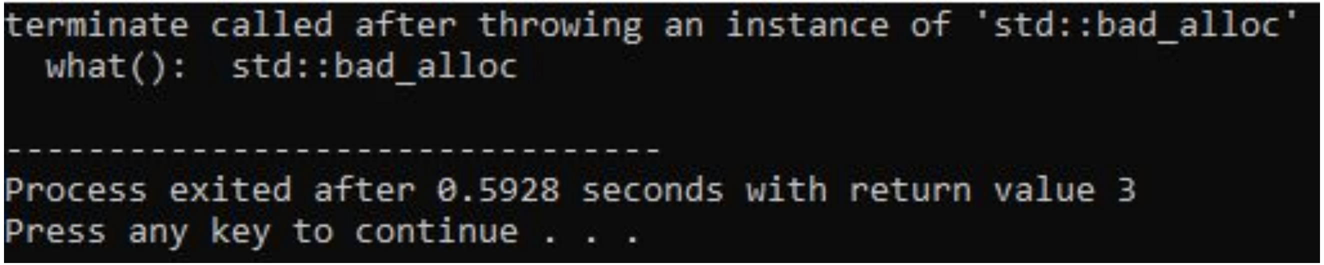
REDEFINING OUR INDEXING

```
#include<stdexcept>
class  DynamicArray {
public:
        int& operator [ ] (int index) {
                try {
                        if ( index < 0 || index >= nextIndex ) {
                                throw out_of_range (“Index Out Of Bounds Exception”);
                        }
                        return *(pa + index);

                } catch (out_of_range &ex) {
                        cout<<ex.what<<endl;
                        return NULL;
                }
        }
};
```

EXCEPTION HANDLING | new

- When we request for dynamic memory allocation from heap memory.
  - Allocation may fail, if there is not sufficient memory available in heap memory.
  - Program crashes with **std::bad_alloc** exception



```
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc


--------------------------------
Process exited after 0.5928 seconds with return value 3
Press any key to continue . . .
```

EXCEPTION HANDLING | new

- Using **try** and **catch block** for memory allocation.

```
try {

    int* pa = new int [size];
    // Allocation successful do some stuff.

} catch (bad_alloc& e) {

    cout<<e.what ( );
    cout<<”Not enough space to define an Array of Length: “<<size;

}
```

EXCEPTION HANDLING | new (nothrow)

- C++ provides another way of handling bad_alloc exception.
  - **nothrow** is a constant used as an argument for new and new [ ] operator.
- nothrow job is to make sure no exception is thrown by new and new [ ]
  - instead a **NULL pointer** is returned.
  - NULL is implicitly converted to false.
  - It simply triggers the overloaded version of new (SOME POLYMORPHISM).

EXCEPTION HANDLING | new (nothrow)

```
int* pa = new (nothrow) int [size];
if ( !pa ) {
      // Allocation failed do something about it.

} else {

    // Allocation successful do some stuff.

}
```

EXCEPTION HANDLING | delete

How does **delete** know how many chunks of memory to delete?

When we define an array with new [ ],

the size is stored in **metadata** on memory location.

delete utilizes that metadata,

which is OS and system dependent.

EXCEPTION HANDLING | delete

- Always avoid throwing exceptions from destructor.
  - When a destructor and copy constructor throws std::terminate is called and program terminates immediately.
- deleting a NULL pointer is safe.
- Typically error arises when trying to delete **same pointer twice** or same **memory from different pointers**.
- We can also fell prey to deleting an **uninitialized pointer**.

EXCEPTION HANDLING | delete

- Always avoid throwing exceptions from destructor.
    - and
- dele
- Typi r sam
- We can also fell prey to deleting an **unitialized pointer**.

Set a pointer to **NULL**

If it is not been initialized yet

OR

you have deleted the memory allocated to the pointer.

LOWER BOUND | >=

- Finding a match in a sorted array.
- Return the pointer to the first element equal to the key.
- If element is not present in the array return the first greater element.
- If no greater element is found as well return the address of the next element of the last element of the array (OUT OF BOUND).
- Should require (Starting location, Ending Location, Key).
- Should return an address.

UPPER BOUND | >

- Finding a match in a sorted array.
- Return the pointer to the first element greater than the key.
- If no greater element is found return the address of the next element of the last element of the array (OUT OF BOUND).
- Should require (Starting location, Ending Location, Key).
- Should return an address.

FINDING A DUPLICATE IN AN UNSORTED ARRAY