Lecture 5

POINTERS & ARRAYS

September 14, 2021

Tuesday

DATA TYPES AND THEIR SIZES

We can find the size of a variable using sizeof operator.

sizeof(char) = 1

sizeof(short) = 2

sizeof(int) = 4

sizeof(float) = 4

sizeof(long) = 4

sizeof(long long) = 8

sizeof(double) = 8

sizeof(long double) = 16

sizeof(ptr) = 8

```
cout<<sizeof (char);
cout<<sizeof (short);
cout<<sizeof (int);
```

POINTERS ARITHMETIC

- Allowed Operations
  - Incremented ++
  - Decremented --
  - Integer Addition + Or +=
  - Integer Subtraction - Or -=
  - One Pointer subtracted from Another

```
ptr + i * sizeof (data type);
```

Why pointer arithmetic is machine and compiler dependent?

POINTERS ARITHMETIC

- Pointer arithmetic is undefined unless performed on an array.
  - we don’t know if two variables of the same type are stored contiguously in memory.

Subtracting two pointers, that do not point to the elements of the array.

ARRAY IN C++

- Arrays in C++ and most programming languages
  - Have to be declared in advance
  - Size have to be known before the program starts.
- Arrays are efficient, provide fast access to a memory location.
  - Memory is allocated at compile time.
  - Memory is allocated from Stack Memory.
- Arrays in C++ have a fixed length and cannot be resized once defined.
  - If the size is too big, space is wasted.
  - If the size is too small, the array overflow with data and program stops.
- What if we don’t know the exact size before hand?

ARRAY IN C++

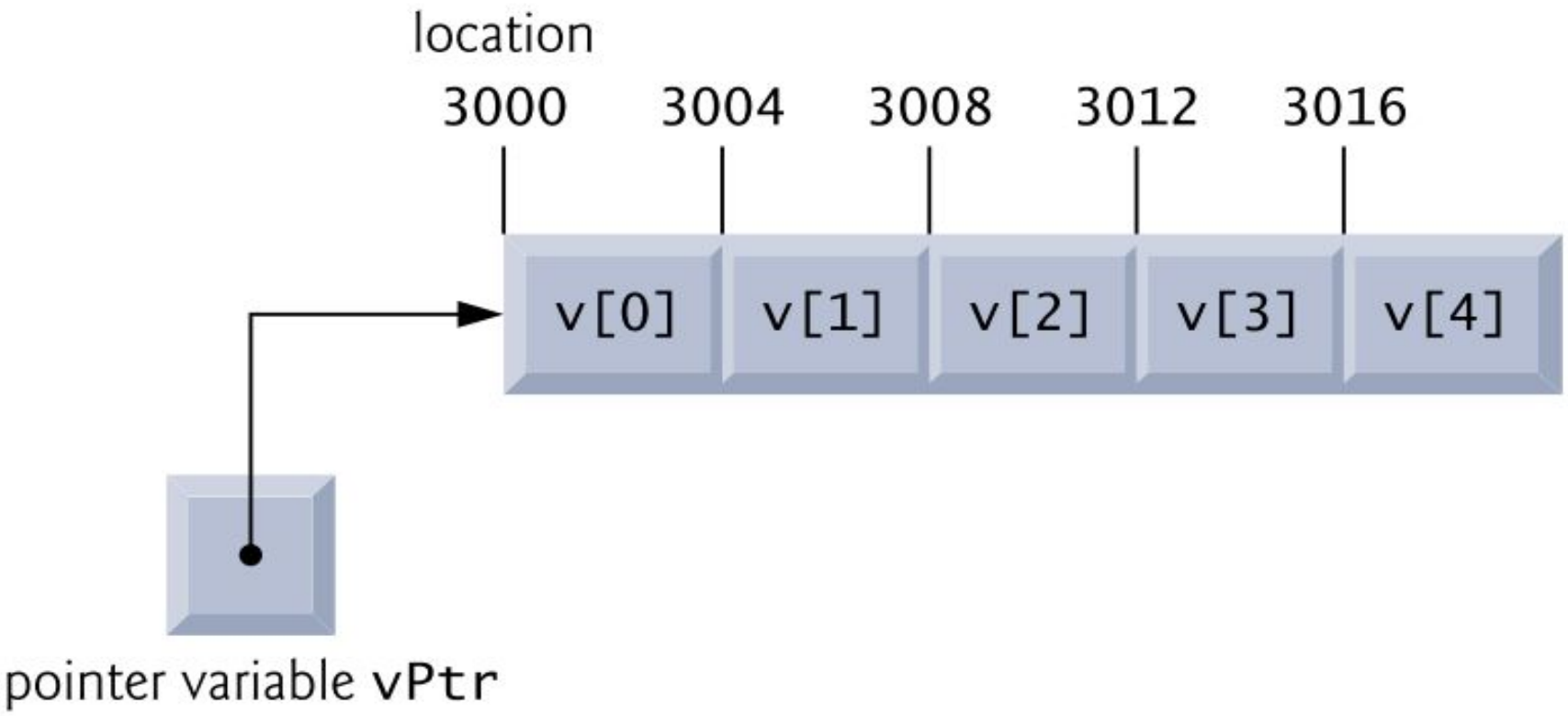
- Name of the arrays is a pointer to the block of memory that can hold 5 integers.
- Array name is fixed and should be considered as constant pointer.

```
// Static Declaration
int arr[5];
int arr [5] = {1, 2, 3, 4, 5};
int [] = {1, 2, 3, 4, 5};
```

```
int v[5], *vPtr;
vPtr = v;          OR  vPtr = &v[0];
```

Modifying array value will be ?

ARRAY IN MEMORY



POINTER NOTATIONS

```
cout<<"Third element: "<<v[2];

for (sum = v [0], i =1; i<5; i++)
        sum += v[i];


//pointer/offset notation (here 2 is the offset to the pointer).
cout<<"The third element: "<< *(vPtr + 2);
cout<<"The third element: "<< *(v + 2);


//pointer/index notation.
cout<<"The third element: "<< vPtr[2];
```

Why Parenthesis?

INDEX BOUNDING

C/C++ does not offer any method or operator to check if the accessed index is valid or not.

- Out of bounds access will result in undefined behavior.
  - May return a garbage value, or modifying a garbage value.
  - Can result in segmentation fault, and crash the application.

SAFE ACCESS

- We want to restrict the user from accessing the invalid memory location.
- Before dereferencing
  - Check if the index is within bounds
    - Yes: proceed with the reading or writing of the memory location.
    - No: generate an error or exception.

DYNAMIC ARRAY

- Flexible Size can grow and shrink at runtime.
  - Requesting the memory allocation with ***new*** keyword.
  - Memory allocation at runtime.
  - Memory is allocated from Heap Memory.
  - After the process is complete releasing the memory is required with ***delete*** keyword.
-

DYNAMIC ARRAY

```
datatype *arrayPointer;
arrayPointer = new datatype [numberOfElements];
arrayPointer [0] = value;
delete [ ] arrayPointer;


int *arrayPointer;
arrayPointer = new int[10];
arrayPointer [0] = 12;
delete [ ] array pointer;
```

CLASS FOR DYNAMIC SAFE ARRAY

```
class  DynamicArray {
        private:
                datatype datamember1;
                datatype datamember2;
                datatype datamember3;
        public:
                datatype memberFunction1 ( );
                datatype memberFunction2 ( );
                datatype memberFunction3 ( );

};
```

RULE OF THREE

- Three member functions which always go together.
  - Copy Constructor
  - Assignment Operator
  - Destructor
- A class with destructor should almost always define the other two.
- A class with Copy Constructor Or Assignment Operator usually defines the other two.

Why Destructor should always be coupled with other two member functions.

CLASS FOR DYNAMIC SAFE ARRAY

- **Data Members**
  - Pointer to the Array
  - Length of the Array
  - Next Index
- **Member Functions**
  - Constructor
  - Destructor
  - Indexing Operation
  - Add a Value
  - Size of Array

CLASS FOR DYNAMIC SAFE ARRAY

```
class  DynamicArray {
        private:
                datatype *pa;
                datatype length;
                datatype nextIndex;
        public:
                DynamicArray ( ) {        }
                ~DynamicArray ( ) {      }
                int& operator [ ] (int index ) {      }
                void add(int val) {        }
                int size() {       }

};
```

DYNAMIC ARRAY | CONSTRUCTOR

```
class  DynamicArray {
            ...........
      public:
            DynamicArray ( ){
                  pa = new int[10];

                  for (int i = 0; i < 10; i++)
                        pa[i] = 0;

                  length = 10;
                  nextIndex = 0;
            }

};
```

DYNAMIC ARRAY | DESTRUCTOR

```
class  DynamicArray {
            ...........
      public:
            ~DynamicArray ( ) {

                  delete [ ] pa;

            }

};
```

DYNAMIC ARRAY | 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);
        }

};
```

DYNAMIC ARRAY | ADD

```
class  DynamicArray {
public:
        void add(int val) {
                int *pnewa;
                if (nextIndex == length) {
                        length = length + 10;
                        pnewa = new int[length];

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

                        for (int j = nextIndex; j < length; j++)
                                pnewa[j] = 0;
                        delete [] pa;
                        pa = pnewa;
                }
                pa[nextIndex++] = val;
        }
};
```

DYNAMIC ARRAY | COPY CONSTRUCTOR

```
class  DynamicArray {
public:
        void add(const DynamicArray& dynamicArray) {

                size = dynamicArray.size;
                length = dynamicArray.length;
                nextIndex = dynamicArra.nextIndex;
                pa = new int [size];
                memcpy(pa, dynamicArray.pa, sizeof(int) * size);

        }
};
```

DYNAMIC ARRAY | ASSIGNMENT OPERATOR

```
class  DynamicArray {
public:
        DynamicArray& operator  = (const DynamicArray& dynamicArray) {

                if(this == dynamicArray)
                        return *this;

                size = dynamicArray.size;
                length = dynamicArray.length;

                int *pnewa = new int [size];
                memcpy(pa, dynamicArray.pa, sizeof(int) * size);

                delete [ ] pa;

                pa = pnewa;

                return *this;
        }
};
```

DYNAMIC ARRAY | SIZE

```
class  DynamicArray {
            ...........
      public:
            int size( ) {
                  return length;
            }


};
```