Lecture 7

TWO DIMENSIONAL DYNAMIC SAFE ARRAYS

September 20, 2021
Monday

REMOVE AN ELEMENT

```
void remove (int index ) {
    try {

        if(index < lower_bound || index > upper_bound)
            throw out_of_range ("Index Out Of Bounds Exception");

        pa[index] = -9999999;
        capacity += 1;
        shrink ( );

    } catch (out_of_range &ex) {

            cout<<ex.what<<endl;
            cout<<"EXITING PROGRAM…";
            exit (EXIT_FAILURE);
    }
}
```

SHRINK THE ARRAY

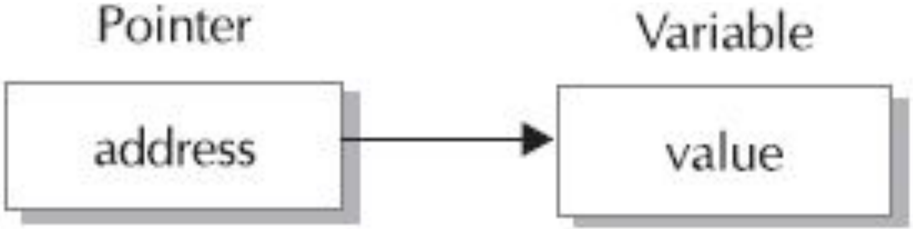
```
void shrink ( ) {
        if (length / capacity <= 2) {
                int* newpa = new(nothrow) int [length - capacity + growth_factor] { };
                if ( !newpa ) {
                        for (int i = lower_bound; i < upper_bound; i++) {
                                if ( pa[i] != -9999999)
                                        newpa[i] = pa[i];
                        }       // Other steps remain the same.
                }else {
                        cout<<"Memory Allocation Failed, closing the application now...";
                        exit ( EXIT_FAILURE );
                }
        }
}
```

exit, EXIT_FAILURE, EXIT_SUCCESS are defined in <stdlib>

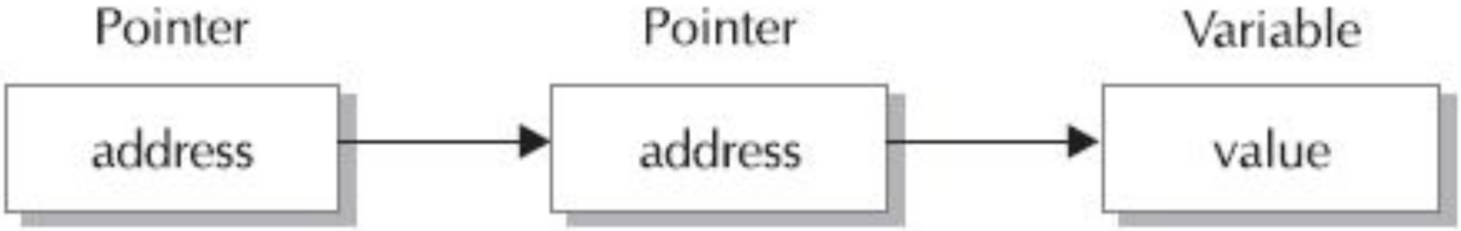
TEMPLATE VERSION

```
template <class T> class Dynarray {
	private:
		T *pa;
		int length;
		int nextIndex;
	public:
		Dynarray();
		~Dynarray();
		T& operator[](int index);
		void add(T val);
		int size();
};
```

DOUBLE POINTERS



Single Indirection



Multiple Indirection

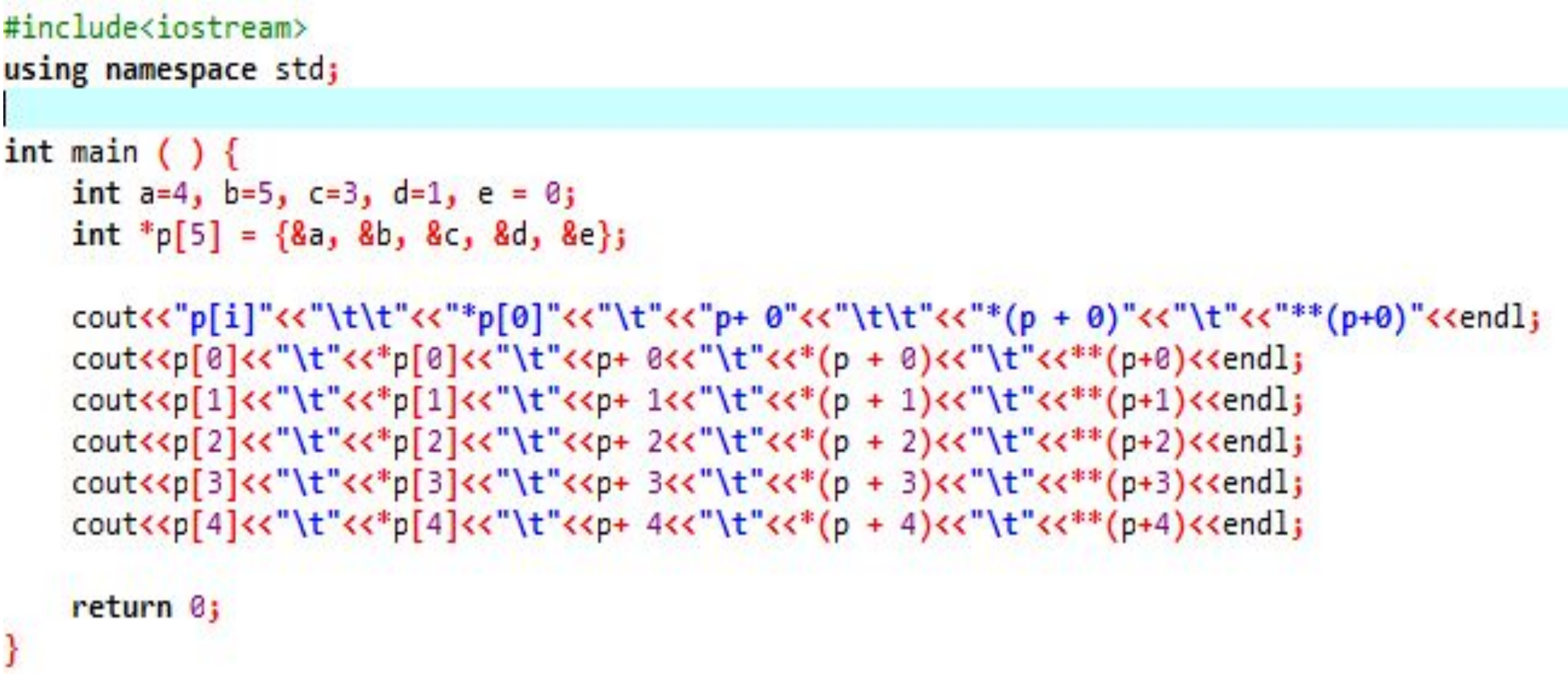
MULTIPLE INDIRECTION

Since a pointer has its own address, we can a have a pointer pointing to another pointer.

- To let compiler know, that this pointer reference to a pointer
- What about Dereferencing (indirection).

```
int num = 10;
int *ptr = &num;
int **pPtr = &ptr;
```

ARRAY OF POINTERS



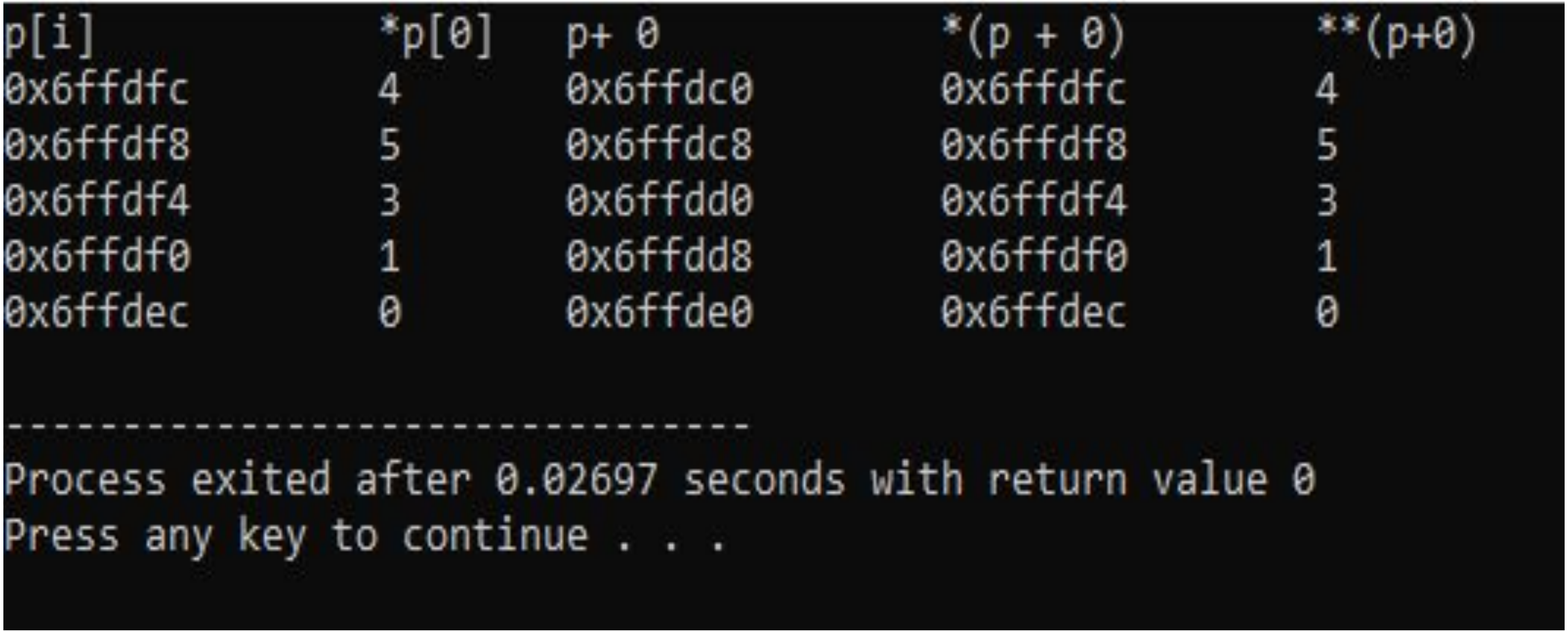
```
#include<iostream>
using namespace std;

int main ( ) {
    int a=4, b=5, c=3, d=1, e = 0;
    int *p[5] = {&a, &b, &c, &d, &e};

    cout<<"p[i]"<<"\t\t"<<"*p[0]"<<"\t"<<"p+ 0"<<"\t\t"<<"*(p + 0)"<<"\t"<<"**(p+0)"<<endl;
    cout<<p[0]<<"\t"<<*p[0]<<"\t"<<p+ 0<<"\t"<<*(p + 0)<<"\t"<<**(p+0)<<endl;
    cout<<p[1]<<"\t"<<*p[1]<<"\t"<<p+ 1<<"\t"<<*(p + 1)<<"\t"<<**(p+1)<<endl;
    cout<<p[2]<<"\t"<<*p[2]<<"\t"<<p+ 2<<"\t"<<*(p + 2)<<"\t"<<**(p+2)<<endl;
    cout<<p[3]<<"\t"<<*p[3]<<"\t"<<p+ 3<<"\t"<<*(p + 3)<<"\t"<<**(p+3)<<endl;
    cout<<p[4]<<"\t"<<*p[4]<<"\t"<<p+ 4<<"\t"<<*(p + 4)<<"\t"<<**(p+4)<<endl;

    return 0;
}
```

ARRAY OF POINTERS



```
p[i]            *p[0]   p+ 0            *(p + 0)        **(p+0)
0x6ffdfc        4       0x6ffdc0        0x6ffdfc        4
0x6ffdf8        5       0x6ffdc8        0x6ffdf8        5
0x6ffdf4        3       0x6ffdd0        0x6ffdf4        3
0x6ffdf0        1       0x6ffdd8        0x6ffdf0        1
0x6ffdec        0       0x6ffde0        0x6ffdec        0


--------------------------------
Process exited after 0.02697 seconds with return value 0
Press any key to continue . . .
```

DYNAMIC 2D ARRAY

We can visualize a 2D array as 1D array where each of its element is another 1D array.

```
int **twoDArray;
twoDArray = new int* [n_rows];

for (int i =0; i < n_rows; i++) {
        twoDArray [i] = new int* [n_cols] { } ;
}
```

CLASS FOR DYNAMIC SAFE ARRAY

```
class  DynamicArray {
    private:
        datatype datamember1;
        datatype datamember2;
        datatype datamember3;

    public:
        datatype memberFunction1 ( );
        datatype memberFunction2 ( );
        datatype memberFunction3 ( );

};
```

CLASS FOR DYNAMIC SAFE ARRAY

- **Data Members**
  - Double Pointer to the Array
  - Number of Rows
  - Number of Columns
  - Next Index
    - Row Number
    - Column Number

- **Member Functions**
  - Constructor
  - Destructor
  - Indexing Operation
  - Add a Value
  - Size of Array
  - Fill Array Values

DYNAMIC ARRAY | CONSTRUCTOR

```
class  DynamicArray {
        private:
                int n_rows, n_cols, **pa2d, row_index, col_index, size;
        public:
                DynamicArray ( int r, int c) n_rows ( r ), n_cols ( c ), pa2d ( NULL),
        row_index ( 0 ), col_index ( 0 ) {
                        pa2d = new int* [n_rows];

                        for (int i = 0; i < n_rows; i++) {
                                pa2d [ i ] = new int [n_cols] { };
                        }

                        size = n_rows * n_cols;
                }
};
```

DYNAMIC ARRAY | DESTRUCTOR

```
class  DynamicArray {
            ………..
      public:
            ~DynamicArray ( ) {
                  if (!pa2d) {

                        for (int i = 0; i< n_rows; i++)
                              delete [ ]  pa2d [ i ];

                        delete pa2d;
                        pa2d = NULL;
                  }
            }
};
```

DYNAMIC ARRAY | COPY CONSTRUCTOR

```
DynamicArray (const DynamicArray& dynamicArray) {

    size = dynamicArray.size;
    n_rows = dynamicArray.n_rows;
    n_rows = dynamicArray.n_cols;
    row_index = dynamicArray.row_index;
    col_index = dynamicArray.col_index;

    pa2d = new int* [n_rows];

    for ( int i = 0; i < n_rows; i++ ) {
        pa2d [ i ] = new int [n_cols];
        memcpy (pa2d [ i ], dynamicArray [ i ], sizeof (int)*n_cols);
    }
}
```

DYNAMIC ARRAY | ASSIGNMENT OPERATOR

```
DynamicArray& operator  = (const DynamicArray& dynamicArray) {

    if(this == dynamicArray)
        return *this;
    for (int i = 0; i < n_rows; i++)
        delete [ ] pa2d [ i ];
    delete  [ ] pa2d;
    pa2d = NULL;

    // Assign data members values JUST LIKE COPY CONSTRUCTOR
    // Copy VALUES JUST LIKE COPY CONSTRUCTOR

    return *this;
}
```

DYNAMIC ARRAY | SIZE

```
class  DynamicArray {
                ………..
        public:
                int size( ) {
                        return n_rows * n_cols;
                }


};
```

REDEFINING OUR INDEXING

```
int& operator ( )  (int row_num, int col_num) {
    try {
        if ( row_num < 0 || row_num >= n_rows || col_number < 0 || col_num >= n_cols )
    {
            throw out_of_range (“Index Out Of Bounds Exception”);
        }
        return pa2d [row_num] [col_num];

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

DYNAMIC SAFE ARRAY | GET

```
int get(int row_num, int col_num) {
        try {
                if ( row_num < 0 || row_num >= n_rows || col_number < 0 || col_num >= n_cols )
        {
                        throw out_of_range (“Index Out Of Bounds Exception”);
                }
                return pa2d [row_num] [col_num];

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

DYNAMIC SAFE ARRAY | SET

```
void set(int row_num, int col_num, int val) {
        // perform the bounds check
        if(!false){
            pa2d [row_num] [col_num] = val;
        }else {
            cout<<”INVALID INDEX”;
        }
}
```

PUSHBACK ROW

```
void pushback_row (int *array, int size) {
        // perform the bounds check
        if(size != n_cols){
            // throw exception
        }else {
            int **newArray = new int*[n_rows + 1];
            for (int i = 0; i <=n_rows; i++){
                newArray[i] = new int [n_cols] { };
                memcpy (newArray [ i ], pa2d [ i ], sizeof (int)*n_cols);
            }
            memcpy(newArray[n_rows + 1], array, sizeof (int)*n_cols);
            // Perform the delete just like constructor
            pa2d = newArray;
        }
}
```

PUSHBACK COLUMN

```
void pushback_col (int *array, int size) {
        // perform the bounds check
        if(size != n_rows){
            // throw exception
        }else {
            int **newArray = new int*[n_rows];
            int j = 0;
            for (int i = 0; i < n_rows; i++) {
                newArray[i] = new int [n_cols + 1] { };

                for (int k = 0; k<=n_cols; k++) {
                    if (k != n_cols) {
                        newArray[i][k] = pa2d[i][k];
                    } else {
                        newArray[i][k] = array[j++];
                    }
                }
            }
```

JAGGED ARRAY

An array of Arrays, where member arrays can be of different length.

- We will need an additional array to store the length of each row.
- Instead of incrementing or decrementing n_cols (variable we used for Dynamic Safe 2D) we will be modifying n_cols [i], which will correspond to its respective row.

JAGGED ARRAY

```
JaggedArray (int r, int row_length [ ] ) : n_rows ( r ), p2ja (NULL),
n_cols ( row_length) {
    p2ja = new int*[n_rows];
    for (int i = 0; i < n_rows; i++) {
        p2ja [ i ]  = new (nothrow) int [ n_cols [ i ] ];
    }
}
```