Lab #10

Dynamic Array

Structured Programming 2017/2018

Agenda

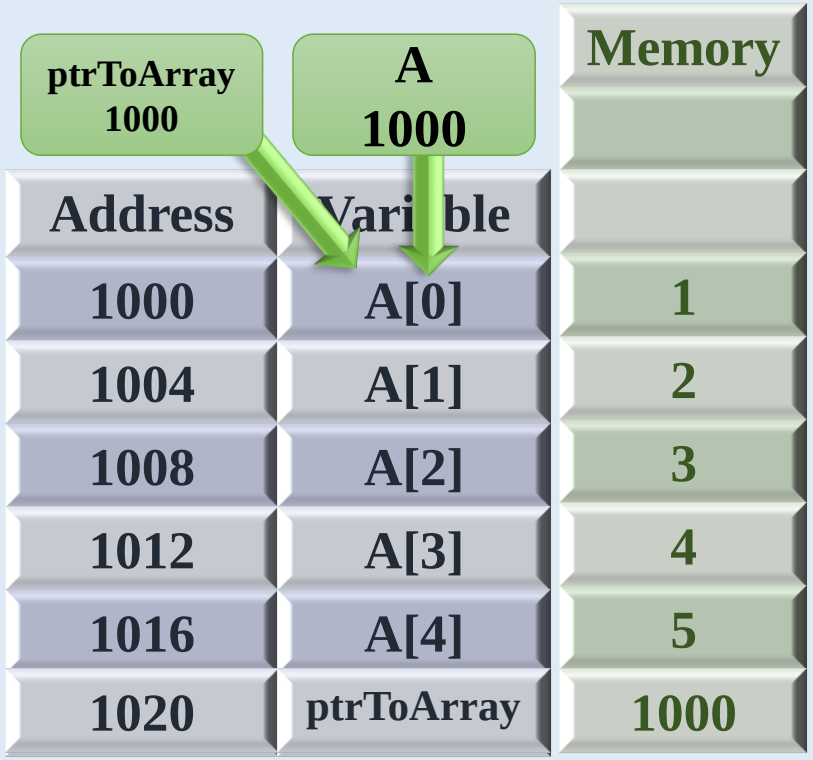
- Pointers and Arrays.
- Dynamic 1D Array.
- Dynamic 2D Array.

Relation between Pointers and Arrays

```
int A[5] = {1, 2, 3, 4, 5};

int *ptrToArray;
```

- The following lines are equivalent:

```
ptrToArray = &A[0];

ptrToArray = A;
```



ptrToArray and A are pointers to an array.
The difference is that A is a constant pointer that cannot be modified, but ptrToArray is a variable that can be modified (by pointing to another location)

Dynamic 1D Arrays

Allocation:

```
int* arr = new int[10];
```

Deallocation:

```
delete [] arr;
```

In function declaration (and definition):

```
Void fun (int * arr int size);
```

||

```
Void fun (int arr[],int size);
```

In function call :

```
int * arr = new int[10];
Fun( arr , 10 );
```

Write a program that asks the user about the **number of elements** he wants to enter, **input the elements** and **store them** in an array. **Show the square** of each element after the input is finished

Dynamic 1D Arrays

```
#include<iostream>
using namespace std;
void main()
{
   int n;
   cout<<"Enter the number of elements"<<endl;
   cin>>n;

   int* A=new int[n];  // allocate n consecutive places in memory.
            // and address of the first is stored in pointer array.

   cout<<"Enter "<<n<<" elements"<<endl;
   for(int i=0; i<n; i++)
        cin>>A[i];

   for(int i=0; i<n; i++)
        cout<<A[i]*A[i];

   delete [] A;  //de-allocation so as not to cause memory leakage
}
```

Hands-on:

Write a function which takes 3 arrays a , b and c of integers each, a is of size n and b of size m. c is an array with m+n integers. The function should put into c the appending of b to a, the first n integers of c from array a, the latter m from b. Then the display c in main.

Input:

Please enter the size of array a : 5

Please enter the size of array b : 3

Please enter the array a :1 2 3 4 5

Please enter the array a : 9 8 7

Output:

Displaying the array c:

1 2 3 4 5 9 8 7

Solution

```
#include<iostream>
using namespace std;
void concat(int *a,int *b,int *c,int n,int m);
int main()
{
   int m,n; //n and m size of array
   cout<<"please enter n and m"<<endl;
   cin>>n>>m;
   int*a=new int[n],*b=new int[m];
   int *c=new int[n+m];
   for (int i = 0; i < n; i++)
      cin>>a[i];
   for (int i = 0; i < m; i++)
      cin>>b[i];
   concat(a,b,c,n,m);
   delete []a;
   delete []b;
   delete []c;
}
```

```
void concat(int *a,int *b,int *c,int n,int m)
{
   for (int i = 0; i < n+m; i++)
   {
      if(i<n)
      {
         c[i]=a[i];
      }
      else
      {
         c[i]=b[i-n];
      }
      cout<<c[i]<<"  ";
   }
}
```

You Must Use Pointers

A vendor is buying from a company several parts, each is represented by:

- The Part number: ***Pnum***
- The Part name: ***Pnam***
- The Quantity the vendor bought from this part: ***PQ***
- The Price of this part: ***PP***

Write a program that reads from the user the number of the parts bought and their data, then calculates how much the vendor should pay for those bought parts.

Hint: the amount the vendor should pay is **Σ PQ * PP** for all the bought parts.

Sample Run

How many Products you want to buy?: 2

Please provide us with 2 products information:

Product# 1:

Name: Bananas

Number: 12345

Price: 5

Quantity: 2

Product# 2:

Name: Tomatoes

Number: 67890

Price: 2

Quantity: 3

The total Price = 16 L.E

```
#include <iostream>
using namespace std;
struct part
{
    int PNum;
    char Pname[10];
    float PPrice;
    int PQuantity;
};
float CalTotalPrice(part * parts , int partsCount);
void main()
{
    int no_of_parts;
    float total=0;
    cout<<"How many Products you want to buy?"<<endl;
    cin>>no_of_parts;
    part * cart= new part[no_of_parts];
```

```
        cout<<"Please Provide us with "<<no_of_parts<<" Products
        information:"<<endl;
        for(int i=0; i<no_of_parts; i++)
        {
                cout<<"Product # "<<i+1<<endl;
                cout<<"Name:";
                cin>>cart[i].PName;
                cout<<endl<<"Number:";
                cin>>cart[i].PNum;
                cout<<endl<<"Price:";
                cin>>cart[i].PPrice;
                cout<<endl<<"Quantity:";
                cin>>cart[i].PQuantity;
                cout<<endl;
        }
        cout<<"You are required to pay “;
        cout<<CalTotalPrice(cart,no_of_parts)<<“ L.E."<<endl;
        delete[] cart;
}
```

```
float CalTotalPrice(part * parts , int partsCount)
{
        float total = 0;
        for(int i=0; i<partsCount; i++)
             total +=parts[i].PPrice *parts[i].PQuantity;
        return total;
}
```

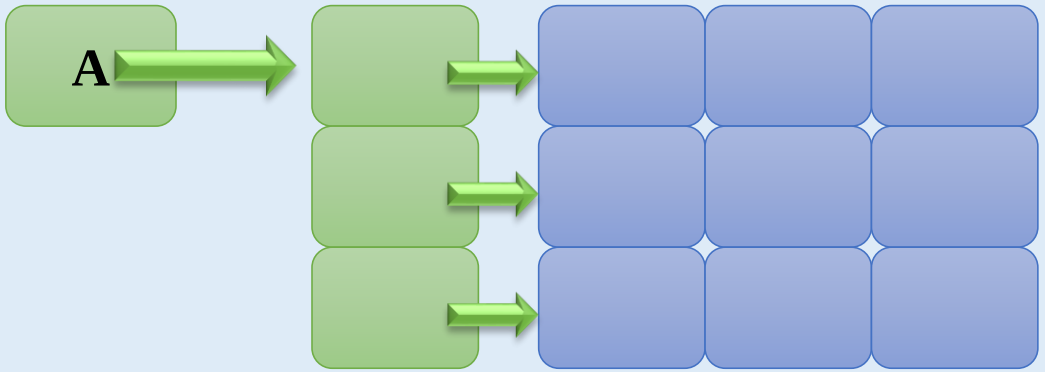
Dynamic 2D Arrays

Allocation:

```
int* *A=new int*[row];

for(i=0; i<row; i++)

  A[i] = new int[col];
```

Deallocation:

```
for(i=0; i<row; i++)

  delete[] A[i];

delete[] A;
```



Dynamic 2D Arrays

In function declaration (and definition):

```
Void fun (int* *arr, int rows , int cols);
 ||
Void fun (int arr[][100] , int rows , int cols);
```

Number of cols must be specified so we set a large number, that makes the first way better than the second

In function call :

```
int* *arr = new int*[rows];
for(int i=0; i<rows; i++)
   arr[i] = new int[cols];
Fun( arr , rows , cols );
```

Write a program to fill in a 2D array from the user and display it in a matrix form.

Dynamic 2D Arrays

```
#include<iostream>
using namespace std;
void main()
{
    int row,col,i;
    cout<<"Enter the number of rows and columns"<<endl;
    cin>>row>>col;

    int* *A=new int*[row];  // create array of pointers
    for(i=0;i<row;i++)
        A[i]=new int[col];

    cout<<"Enter "<<row*col<<" Element"<<endl;
    for(i=0;i<row;i++)
        for(int j=0;j<col;j++)
            cin>>A[i][j];
```

Dynamic 2D Arrays

```
    cout<<"The elements are"<<endl;
    for(i=0;i<row;i++)
    {
       for(int j=0;j<col;j++)
            cout<<A[i][j]<<" ";
       cout<<endl;
    }

    // deallocate memory
    for(i=0;i<row;i++)
       delete[] A[i];
    delete[] A;
}
```

Hands-on :

Write a program to fill in a 2D array then ask the user to update a specific row or column (zero all the elements of that row or column). and then it displays the modified matrix.

Solution:

```
#include<iostream>
using namespace std;
void main()
{
    int row,col,i;
    cout<<"Enter the number of rows and columns"<<endl;
    cin>>row>>col;

    int* *A=new int*[row];  // create array of pointers
    for(i=0;i<row;i++)
        A[i]=new int[col];

    cout<<"Enter "<<row*col<<" Element"<<endl;
    for(i=0;i<row;i++)
        for(int j=0;j<col;j++)
            cin>>A[i][j];
```

```
char choice;
int r,c;
cout<<"You want to modify r or c?"<<endl;
cin>>choice;
While(1)
{
    if(choice=='r' || choice=='R')
    {
        cout<<"Row#:";
        cin>>r;
        if(r <= row)
            for(int i=0; i<col; i++)
                A[r-1][i]=0;
        else
                cout<<"Invalid!"<<endl;
        break;
    }
```

```
else if(choice =='c' || choice=='C')
 {
     cout<<"Col#:";
     cin>>c;
     if(c<=col)
        for(int i=0;i<row;i++)
            A[i][c-1]=0;
     else
            cout<<"Invalid!!"<<endl;
     break;
 }
 else
 {
     cout<<"Invalid choice!!"<<endl;
     cout<<"You want to modify r or c?"<<endl;
     cin>>choice;
 }
}
```

```
    cout<<"The elements are"<<endl;
    for(i=0;i<row;i++)
    {
        for(int j=0;j<col;j++)
              cout<<A[i][j]<<" ";
        cout<<endl;
    }

    // deallocate memory
    for(i=0;i<row;i++)
        delete[] A[i];
    delete[] A;
}
```

Thank you ☺