Lab-2

(Pointers and Dynamic Memory allocation)

Data Structures

Khalid Mengal

Contents

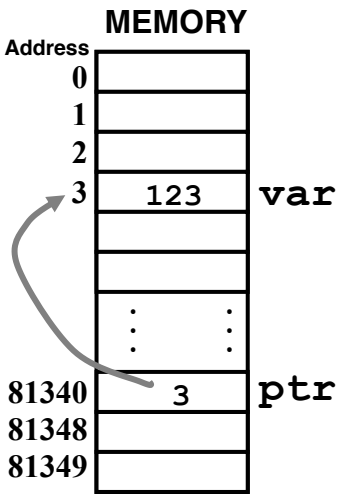
- Pointers
- Dynamic variables
- new and delete operator
- Pointers and functions
- Pass by Reference
- Exercise

Pointers

- A pointer is a variable that holds the *address* of something else.

```
int var;
int *ptr;

var = 123;
ptr = &var;
```



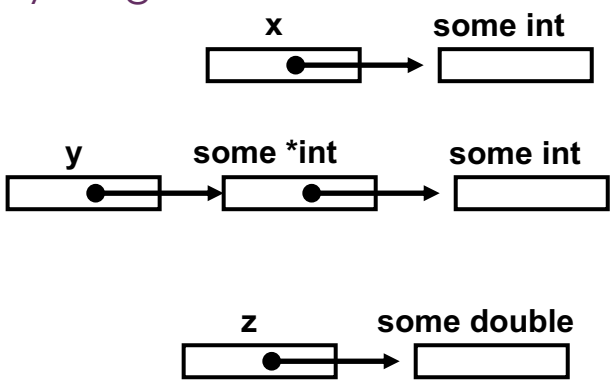
3

Advantages of using Pointers

- Provide direct access to memory
- Make the program simple and efficient
- Allocate / deallocate memory during the execution of the program
- Pass arrays and strings to functions
- Return more than one value from a function

Pointers to anything

```
int *x;
int **y;
double *z;
```



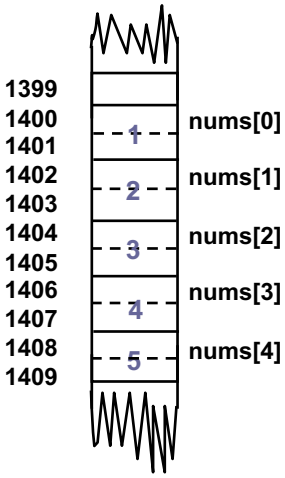
5

Array Notation

There is a close association between pointers and arrays.
An array name is basically a *const* pointer.

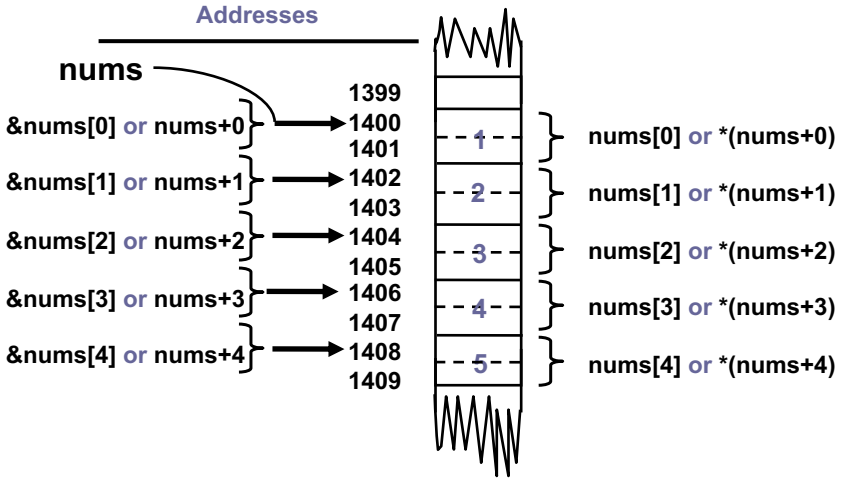
```
short nums[ ]= { 1, 2, 3, 4, 5 };

for(int index=0; index<5; index++)
    cout<< nums[index];
```



6

Pointer Notation



7

Array Notation vs Pointer Notation

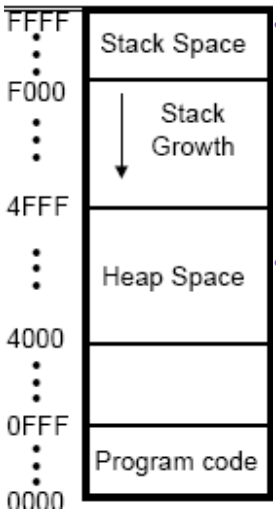
Array Notation

```
int nums[ ]= { 92, 81, 70, 69, 58 };
for(int index=0; index<5; index++)
    cout<< nums[index];
```

Pointer Notation

```
int nums[ ]= { 92, 81, 70, 69, 58 };
for(int index=0; index<5; index++)
    cout<< *(nums+index);
```

Program Address Space



- A program's ***address space*** is the range of addresses a program can operate on.
  - A 64-bit machine provides a program with a 64-bit logical address space.
    - Note that the logical address space is not the same as physical memory addresses.
- A program address space is divided into three main areas:
  1) **Code area** - near start of space
  2) **Heap** - middle of address space
  3) **Stack** - near top of address space

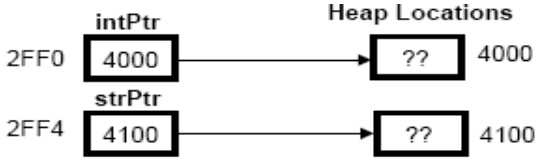
  stack grows, but direction of stack growth is OS dependent

9

Allocating Variables Using new

- **new** operator can be used to allocate dynamic memory in the heap
  - ***variableType *variableName* = new *variableType*();**
- For example:

```
int *intPtr = new int();      // On same line
string *strPtr;               // Two lines
strPtr = new string();
```



Lab-1

10

The delete Operator

- Unlike Java, you as a programmer in C++ are responsible for deleting any dynamic memory objects you create.
- Deletion is accomplished by using the **delete** operator and passing the pointer to the object to be deleted:

```
delete ptrName;          // Single-object version
delete[ ] arrayPtrName;  // Array version
```

- For example:

```
int *intArray = new int[35];
double *dPtr = new double;   // () are optional  for base types
delete[] intArray;           // Delete array
delete dPtr;                  // Delete double
```

11

Pointers & Functions: **swap()**

```
void swap(int *p, int *q)
 {
    int tmp;
    tmp = *p;
    *p = *q;
    *q = tmp;
 }
 .
 .
 .
 swap(&a, &b)          //Call swap function
```

12

Pass By Reference

```
void swap(int &p, int &q)
 {
    int tmp;
    tmp = p;
    p = q;
    q = tmp;
 }
 .
 .
 .
 swap(a, b)              //Call swap function
```

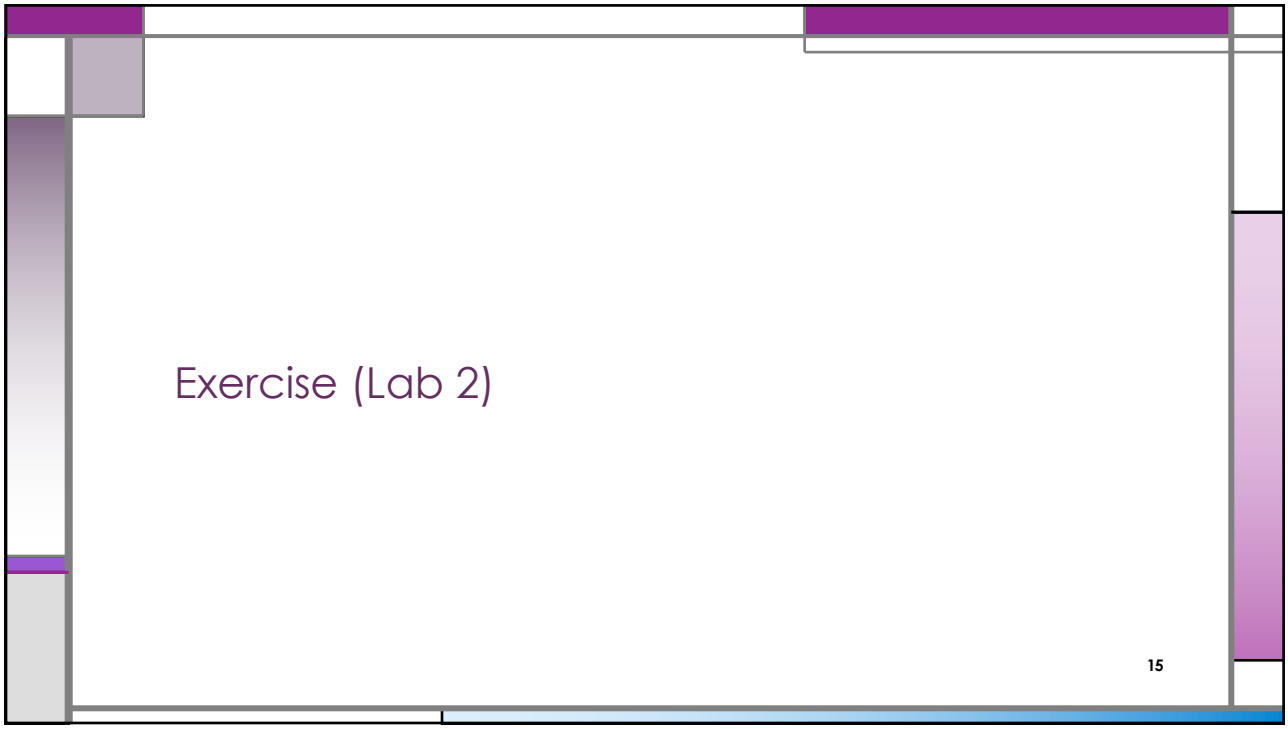
Random Number in C++

- The **rand**() function computes a sequence of pseudo-random integers in the range of 0 to RAND_MAX

```
int number = rand();    //number will be assigned a value between 0-RAND_MAX
int number = rand() % 101;  //number will get a value between 0 and 100
```

- **srand**(arg) function sets its argument as the seed for a new sequence of pseudo-random numbers returned by **rand**().

```
srand(time(NULL));
```


Exercise (Lab 2)
15