# EECS168/169-Lab5

University of Kansas

# Pointers Intro – "&": Address-of operator

- Can we know the "address in memory" of a variable?
- Yes! By using the "&" operator

```
int num = 10;
cout << num;        //Output> 10
cout << &num;       //Output> 0x7ffe3038558c
```

- Variables which hold address of other variables are called Pointers!
- Variable store → value (int, char, double, float)
- Pointer store → address(int, char, double, float)

# Pointers Intro – "*" : Dereference operator

- Previous example:

```
int num = 10;
cout << num;        //Output> 10
cout << &num;       //Output> 0x7ffe3038558c
```

- **Defining pointer variables:**

```
int *p;          //Define int pointer variable p
p = &num;        //Initialize it to address of num
cout << p;       //Output> 0x7ffe3038558c
```

# Arrays

- C++ provides a data structure, **the array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

# Arrays

```
int num1,num2,num3;
num1=5;num2=6;num3=7;
```

Array syntax :

**Data-type arrayname [arraysize];**

**`int array_num[3]; // Define an array of size 3 holding integers`**

**`//array created on call-stack`**

Once an array is created it's size cannot change.

**Array index starts from 0. For this array indices are: 0,1,2**

```
array_num[0]=8;
array_num[1]=9;
array_num[2]=10;

cout << array_num[0]; //Ouput> 8
cout << array_num[1]; //Ouput> 9
cout << array_num[2]; //Ouput> 10
```

**You can read beyond array size but result is unpredictable!**

```
cout << array_num[3]; //Ouput> 0(value outside of array size)
```

# Arrays

**DISPLAY 7.2 An Array in Memory**

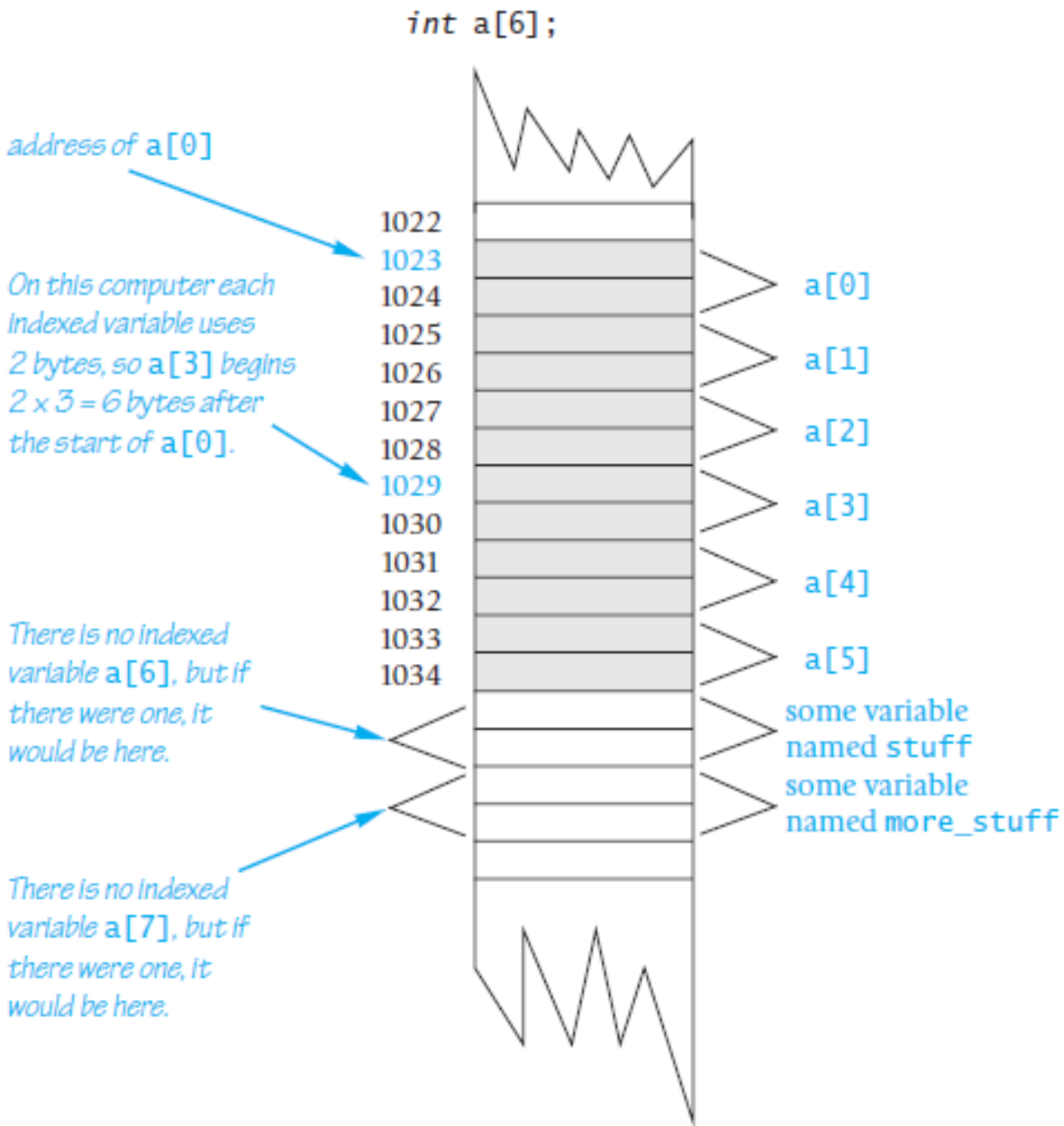



Fig: Problem Solving with C++, Walter Savitch, 9e

# Arrays – Dynamically defined

## Non-dynamic array or static array

```
int array_num[3];


array_num[0]=8;
```

## Dynamically created array

```
int *array_nums = nullpltr;   // not referring to an array yet
array_nums = new int[3];      // Dynamic array - heap allocated


array_nums[0]=8; array_nums[1]=9; array_nums[2]=10;
```

Heap-allocated arrays must be deleted at the end of main. Stack-allocated arrays are automatically deleted.

```
delete[] array_nums;
```

# File operations – stream objects

Open files stored in disk from your program for reading and writing using **streams**

**Stream**: Variable called ***object***. It has **two types** for input and output:

| | | |
|---|---|---|
| **Input** | **ifstream** | **i**nput **f**ile **stream** |
| **Output** | **ofstream** | **o**utput **f**ile **stream** |

**Eg**: To open a file called **infile.txt** present in disk & write to a file **outfile.txt** also in disk

```
#include <fstream> //Include the fstream library
#include <iostream>

ifstream in_stream;                  // declare a variable of stream type - input
in_stream.open("infile.txt");        // Let infile.txt point to in_stream variable
int one_number, another_number;
in_stream >> one_number >> another_number; // Read from file via in_stream. Similar to cin >>

ofstream out_stream;                 // declare a variable of stream type – output
out_stream.open("outfile.txt");      // Let outfile.txt point to out_stream variable
out_stream << "one_number = " << one_number << " another_number = " << another_number; // Write. Similar to cout <<


in_stream.close( );                  // close file connected to in_stream
out_stream.close( );                 // close file connected to out_stream
```

# File operations – file names

**Two names** for single file used in program:

**1. Real file name –** external file name. Eg: infile.txt, outfile.txt etc

Used only **ONCE** while opening the file in program.

External file names

```
in_stream.open("infile.txt");
out_stream.open("outfile.txt");
```

**2. Stream name –** internal file name. Eg: variable name given to type ifstream and ofstream

```
ifstream in_stream;
ofstream out_stream;
```

Used for reading/writing in **program statements**.

Internal file names

```
in_stream >> one_number >> another_number; // Read
out_stream << one_number << another_number; // Write
```

# File operations

```
//Include the fstream library
#include <fstream>
#include <iostream>
int main(int argc, char **argv)
{
  int x=0;  int y=0;   int z=0;
  std::ifstream inFile; //create a variable of type std::ifstream. It will enable reading from files.
  inFile.open("someFile.txt"); //open a file that has values in it in same dir

  inFile >> x; //Read a value from the file and put it in the variable x
  inFile >> y; //Read a value from the file and put it in the variable y
  inFile >> z; //Read a value from the file and put it in the variable z

  std::cout << "x = " << x << '\n';
  std::cout << "y = " << y << '\n';
  std::cout << "z = " << z << '\n';
  return(0);
}
```

# Exercise1 : Basic Numeric Computation

- Create an array of 5 doubles. **(We know the size at compile time, where do you want to allocate this?)**
- Now that you have an array, we need to get values. Create a loop that will ask the user to input values for the array. Display the values back to the user after you obtain all of them. Test this before moving on.
- Calculate sum, average, min, and max for the array of doubles.

```
Please enter 5 numbers
Input a number into your array: 10.0
Input a number into your array: 6.0
Input a number into your array: 4.0
Input a number into your array: 0.0
Input a number into your array: 15.0
Here are all the numbers in your array:
10.0 6.0 4.0 0.0 15.0
The sum of all the values is: 35.0
The average of all the values is: 7.0
The largest value is : 15.0
The smallest value is : 0.0
```

# Exercise2 : Array of strings

- Create an array of strings. Let the user decide how big this array is, but it must have at least 1 element. Prompt them until them give a valid size.
- Prompt the user to populate the array with strings
- Display the longest and shortest string.

```
Input an array size for you words array: 5
Now please enter 5 words
Input a word: apples
Input a word: eat
Input a word: banana
Input a word: spectacular
Input a word: no
The longest word is : spectacular
The shortest word is : no
```

# Exercise3 : File Reading Part 1

```
$ gedit input.txt &
```

Copy the example contents into input.txt:

```
5
105
15
20
35
47
```

How many doubles in input.txt

Read from file first value as int.

Create an array to store the numbers.

# Exercise 3 (cont.)

After you've read in the values display them to the screen in the following format:

```
Contents of input.txt:
[105, 15, 20, 35, 47]

Input a value to search for:
```

The search prompt obtains a value from the user and confirms whether or not it is in the array

Example:

```
Contents of input.txt:
[105, 15, 20, 35, 47]

Input a value to search for: 5
5 is not in the array.
Do you wish to quit (y/n): n
Input a value to search for: 105
105 is in the array.
Do you wish to quit (y/n): y
```

Let the user search as many times as they want.

# Exercise4

- Create a program that performs the following:
- Prompts the user for a file name and stores it
  - Prompt the user until they give the name of a file that can be opened
- Read in and store the double in the file then...
- Create a copy of the values in another array of the same size
  - Then, normalize the values in the copy (yes this will change values)
  - The largest value is always normalized to 1.0 and the smallest to 0.0
  - The values in between are adjusted to be between 0 and 1 but represent their former ratios to the original number
  - Hint: (value - min) / (max - min)
- Create a copy of the original array then reverse the order of value (e.g. the first value in the original will be the last value in the new array)
- Once you have the normalized and reversed arrays, store them in two separate files along with the original values
  - Store the normalized array in a file called "normalized.txt"
  - Store the reversed array in a file called "reversed.txt"

```
Original array: [10.0, 40.0, 20.0, 30.0, 50.0]
Normalized array: [0, .75, .25, .5, 1]
Reversed array: [50.0, 30.0, 20.0, 40.0, 10.0]
```

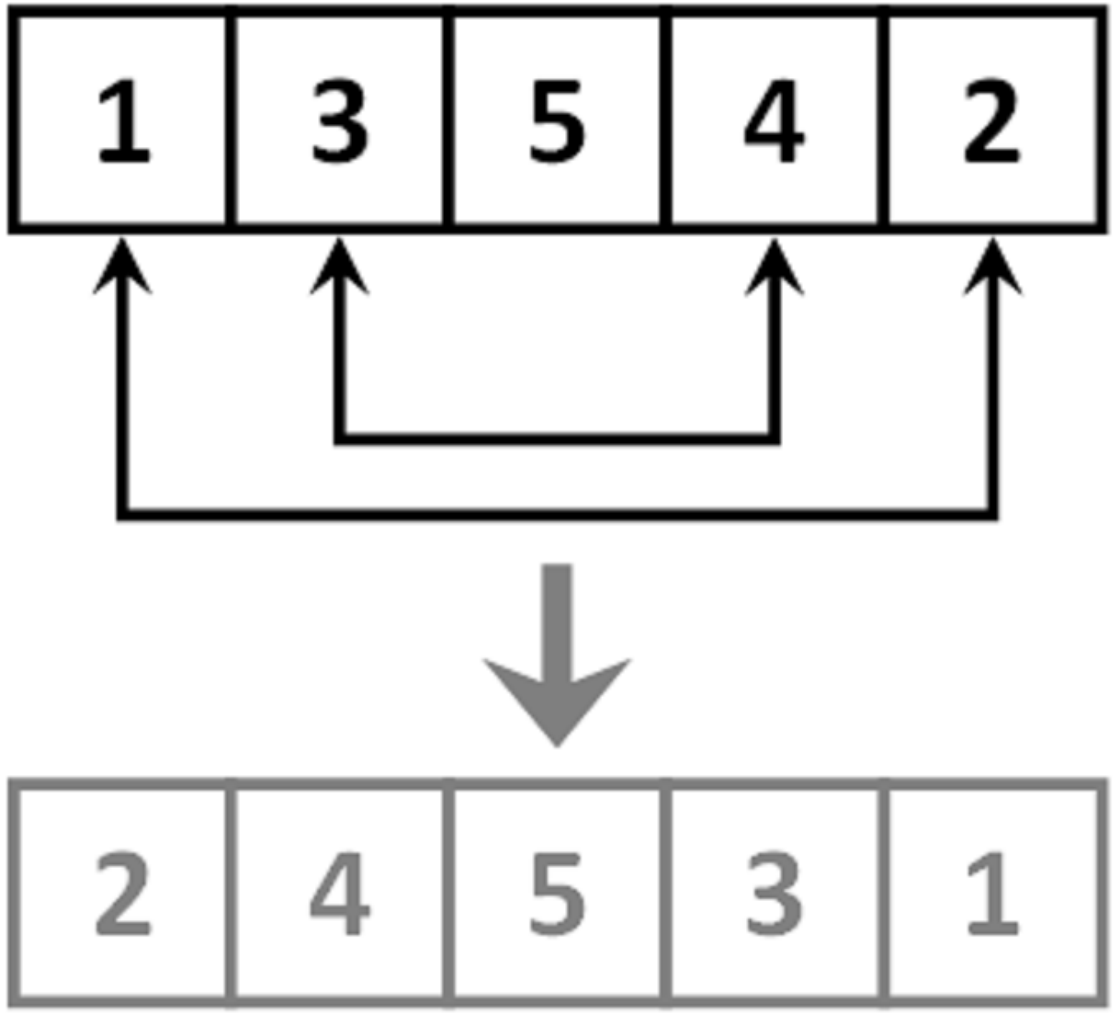
# Exercise4 : File Reading : Normalizing

```
(value - min) / (max - min)
```

```
Original array: [10.0, 40.0, 20.0, 30.0, 50.0]
Normalized array: [0, .75, .25, .5, 1]
```

```
(value - min) / (max - min) = (10.0 - 10.0)/(50.0 - 10.0) =  0
(value - min) / (max - min) = (40.0 - 10.0)/(50.0 - 10.0) =  30.0/40.0 = 0.75
(value - min) / (max - min) = (20.0 - 10.0)/(50.0 - 10.0) =  10.0/40.0 = 0.25
(value - min) / (max - min) = (30.0 - 10.0)/(50.0 - 10.0) =  20.0/40.0 = 0.5
(value - min) / (max - min) = (50.0 - 10.0)/(50.0 - 10.0) =  40.0/40.0 = 1
```

# Exercise4 : File Reading : Reversing array




Ref: http://javarevisited.blogspot.com/2015/03/how-to-reverse-array-in-place-in-java.html

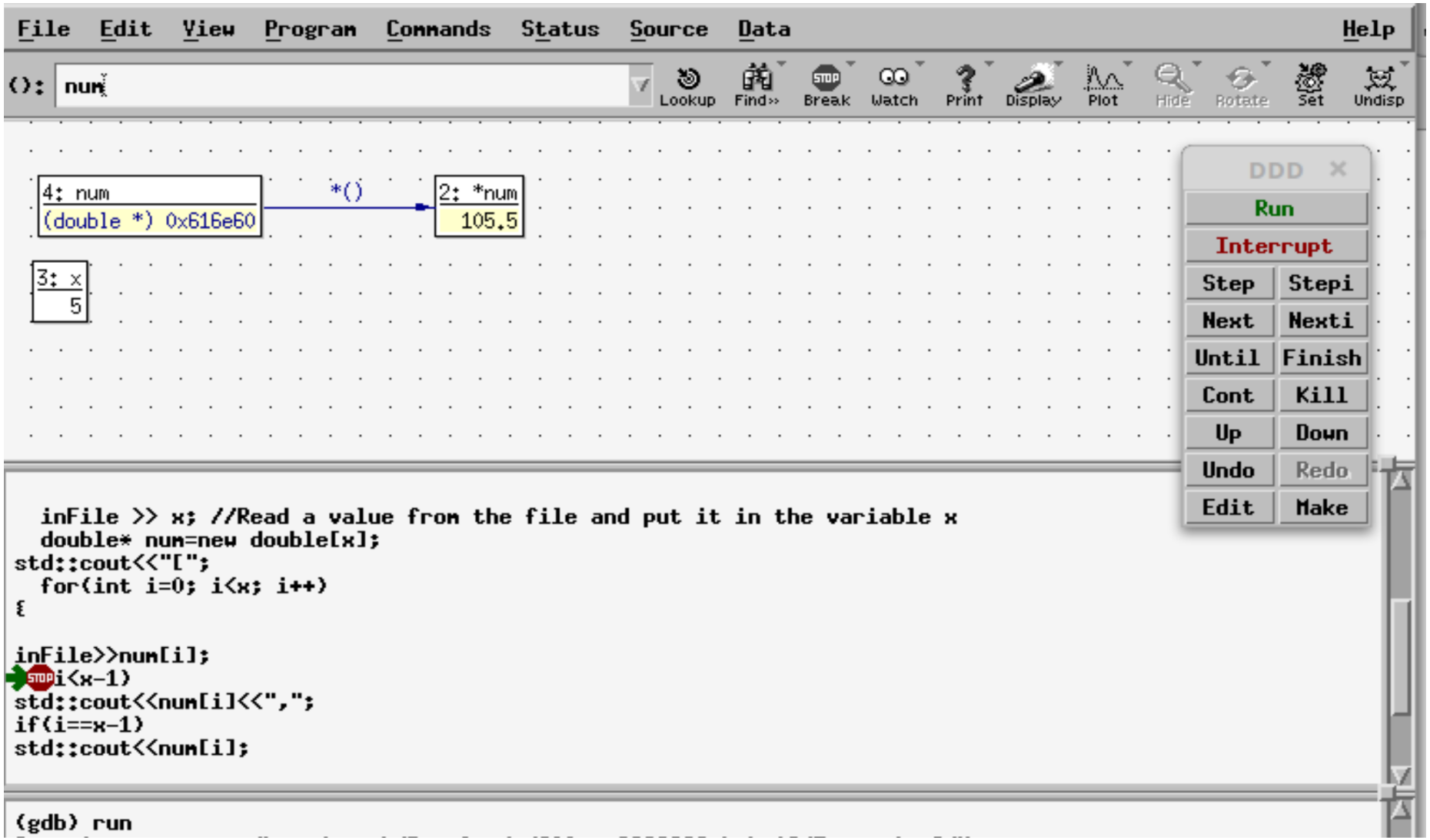
# Warning!

- We will test your program with different files that may have different contents.
- Test your program by changing the contents of your file and run your program again
- You shouldn't need to recompile your code even if the input file changes changes
- You can assume the file will be properly formatted and contain good data

# Exercise for169 Only

- You have all the requirements as the 168, but with the additional complication of keep your array sorted at all times. You don't need to use a sorting algorithms (e.g. bubble, insertion, or quick sort), but as your are getting values from the user or file you will place the value in the index that keeps the array sorted. You may have to shift values behind the newly added value down.
- The array can be in ascending or descending order to begin with and when it is reversed it will be in the opposite order.

# Debugger Review



# Memory Leaks

`delete` is not called for the `new`.

```
$ valgrind --leak-check=full ./YourProgram
```

**No memory leak:**

```
==13654== HEAP SUMMARY:
==13654==     in use at exit: 0 bytes in 0 blocks
==13654==   total heap usage: 1 allocs, 1 frees, 12 bytes allocated
==13654==
==13654== All heap blocks were freed -- no leaks are possible
==13654== For counts of detected and suppressed errors, rerun with: -v
```

# Memory Leaks

`delete` is not called for the `new`.

```
$ valgrind --leak-check=full ./BigArrayProgram
```

**With memory leak:**

```
==14163== HEAP SUMMARY:
==14163==     in use at exit: 12 bytes in 1 blocks
==14163==   total heap usage: 1 allocs, 0 frees, 12 bytes allocated
==14163==
==14163== 12 bytes in 1 blocks are definitely lost in loss record 1 of 1
==14163==    at 0x4A0700A: operator new[](unsigned long) (in
/usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==14163==    by 0x400B43: main (arrays.cpp:40)
==14163==
==14163== LEAK SUMMARY:
==14163==    definitely lost: 12 bytes in 1 blocks
==14163==    indirectly lost: 0 bytes in 0 blocks
==14163==      possibly lost: 0 bytes in 0 blocks
==14163==    still reachable: 0 bytes in 0 blocks
==14163==         suppressed: 0 bytes in 0 blocks
==14163== For counts of detected and suppressed errors, rerun with: -v
```

# Lab Cycle Issues

- If you have alloc and free = 0, and a leak summary with 0 for everything, Great!
- If you have allocs or frees != 0 but in the leak summary you have:

```
==14163== LEAK SUMMARY:
==14163==    definitely lost: 0 bytes in 0 blocks
==14163==    indirectly lost: 0 bytes in 0 blocks
==14163==      possibly lost: 0 bytes in 0 blocks
==14163==    still reachable: 0 bytes in 0 blocks
==14163==         suppressed: 0 bytes in 0 blocks
```

**Your code is acceptable!**

**Anything else is a memory leak → alloc/free != 0 and leak summary with lost bytes**