C++ Programming

Instructor: Rita Kuo
Office: CS 520E
Phone: Ext. 4405
E-mail: rita.kuo@uvu.edu

Mapping zyBooks Chapters

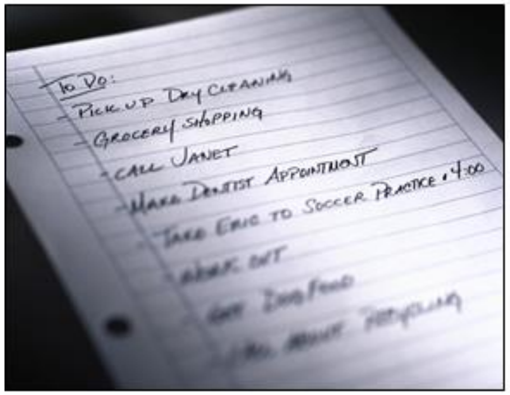
Topics on the slides	Chapters in zyBooks
Arrays	5.1, 5.11, 5.12
Character Arrays/C Strings	5.18
C String Library	5.19
String Library	5.21
Vector	5.2, 5.4, 5.6, 5.7
Arrays vs. Vectors	5.17
Multidimensional Arrays	5.16

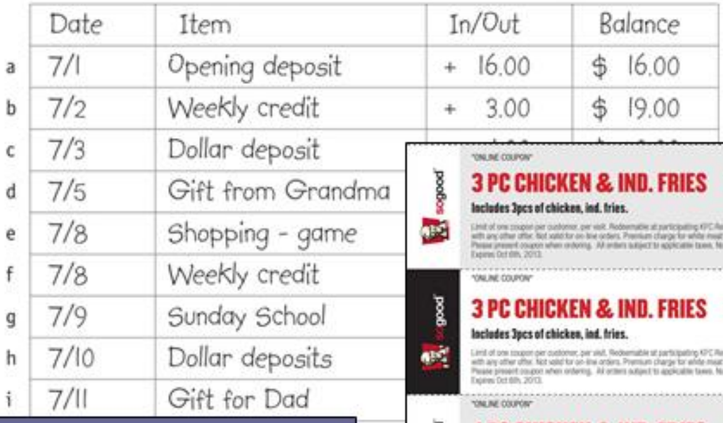
Self-study Chapters: 5.13, 5.14, 5.15, 5.20, 5.5, 5.8, 5.9, 5.10

Arrays and Vectors

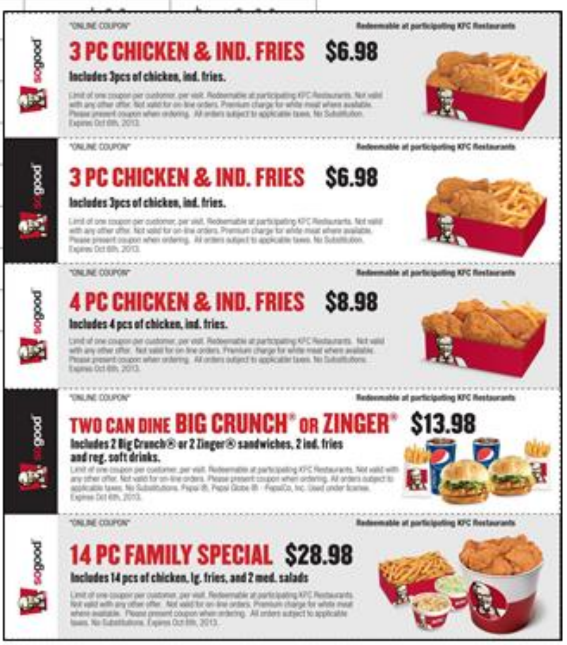
List in Real Life

- It is often the case in computer science that you will process a list of items


To Do:
- Pick up Dry Cleaning
- Grocery Shopping
- Call Janet
- Make Dentist Appointment
- Take Eric to Soccer Practice · 4:00
- Work out
- Get Dog Food
- [illegible]



	Date	Item	In/Out	Balance
a	7/1	Opening deposit	+ 16.00	$ 16.00
b	7/2	Weekly credit	+ 3.00	$ 19.00
c	7/3	Dollar deposit		
d	7/5	Gift from Grandma		
e	7/8	Shopping - game		
f	7/8	Weekly credit		
g	7/9	Sunday School		
h	7/10	Dollar deposits		
i	7/11	Gift for Dad		


"ONLINE COUPON" Redeemable at participating KFC Restaurants

3 PC CHICKEN & IND. FRIES $6.98
Includes 3pcs of chicken, ind. fries.
Limit of one coupon per customer, per visit. Redeemable at participating KFC Restaurants. Not valid with any other offer. Not valid for on-line orders. Premium charge for white meat where available. Please present coupon when ordering. All orders subject to applicable taxes. No Substitution. Expires Oct 6th, 2013.

"ONLINE COUPON" Redeemable at participating KFC Restaurants

3 PC CHICKEN & IND. FRIES $6.98
Includes 3pcs of chicken, ind. fries.
Limit of one coupon per customer, per visit. Redeemable at participating KFC Restaurants. Not valid with any other offer. Not valid for on-line orders. Premium charge for white meat where available. Please present coupon when ordering. All orders subject to applicable taxes. No Substitution. Expires Oct 6th, 2013.

"ONLINE COUPON" Redeemable at participating KFC Restaurants

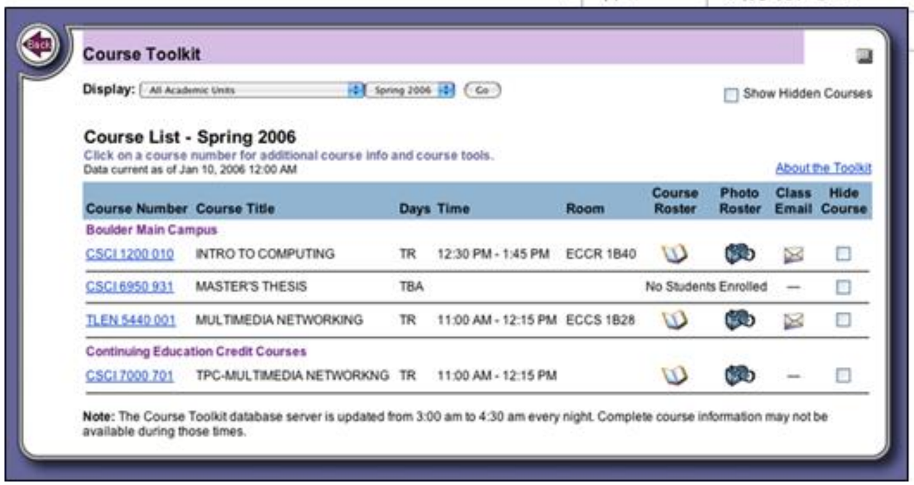
4 PC CHICKEN & IND. FRIES $8.98
Includes 4 pcs of chicken, ind. fries.
Limit of one coupon per customer, per visit. Redeemable at participating KFC Restaurants. Not valid with any other offer. Not valid for on-line orders. Premium charge for white meat where available. Please present coupon when ordering. All orders subject to applicable taxes. No Substitutions. Expires Oct 6th, 2013.

"ONLINE COUPON" Redeemable at participating KFC Restaurants

TWO CAN DINE BIG CRUNCH® OR ZINGER® $13.98
Includes 2 Big Crunch® or 2 Zinger® sandwiches, 2 ind. fries and reg. soft drinks.
Limit of one coupon per customer, per visit. Redeemable at participating KFC Restaurants. Not valid with any other offer. Not valid for on-line orders. Please present coupon when ordering. All orders subject to applicable taxes. No Substitutions. Pepsi ®, Pepsi Globe ® - PepsiCo, Inc. Used under license. Expires Oct 6th, 2013.

"ONLINE COUPON" Redeemable at participating KFC Restaurants

14 PC FAMILY SPECIAL $28.98
Includes 14 pcs of chicken, lg. fries, and 2 med. salads
Limit of one coupon per customer, per visit. Redeemable at participating KFC Restaurants. Not valid with any other offer. Not valid for on-line orders. Premium charge for white meat where available. Please present coupon when ordering. All orders subject to applicable taxes. No Substitutions. Expires Oct 6th, 2013.



Course List - Spring 2006
Click on a course number for additional course info and course tools.
Data current as of Jan 10, 2006 12:00 AM
About the Toolkit

Course Number	Course Title	Days	Time	Room	Course Roster	Photo Roster	Class Email	Hide Course
Boulder Main Campus								
CSCI 1200 010	INTRO TO COMPUTING	TR	12:30 PM - 1:45 PM	ECCR 1B40	📖	●	✉	☐
CSCI 6950 931	MASTER'S THESIS	TBA			No Students Enrolled		—	☐
TLEN 5440 001	MULTIMEDIA NETWORKING	TR	11:00 AM - 12:15 PM	ECCS 1B28	📖	●	✉	☐
Continuing Education Credit Courses								
CSCI 7000 701	TPC-MULTIMEDIA NETWORKNG	TR	11:00 AM - 12:15 PM		📖	●	—	☐

Note: The Course Toolkit database server is updated from 3:00 am to 4:30 am every night. Complete course information may not be available during those times.

List

- List differ from set in that sets do not allow duplicates
  - E.g., $A = \{0, 3, 4, 5, 6, 3, 4, 5\}$ is a list (3, 4, and 5 are repeated)
    $B = \{0, 3, 4, 5, 6\}$ is a set
  - However, a set is a list
  - In math, lists are known as multisets
- Mathematically, you can think of a list as a sequence:
  - $a_1, a_2, a_3, \ldots, a_n$
  - Where n is some integer and a_i is an integer, or a real number, or a string, etc. (whatever type your list consists of)
  - a_i references the i-th element in the list. The $_i$ is know as a subscript
  - For example, assume n = 1000 and a_i are natural numbers. The sequence you could generate is:
    $1, 2, 3, 4, 5, \ldots, 998, 999, 1000$

Basic List in C/C++

- A homogeneous collection of items (e.g., integers, characters, strings, real numbers, etc) that allows for duplicates
- Represent 1, 2, 3, 4, 5, …, 998, 999, 1000 in C
  - You could define a 1000 different variables. For example

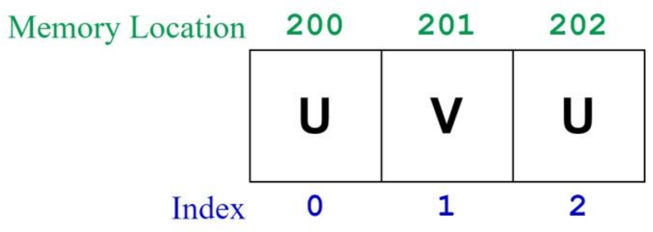
```
int a1 = 1, a2 = 2, a3 = 3, a4 = 4, a5 = 6;
/* declare variables a6 through a1000 */
int a998 = 998 , a999 = 999 , a1000 = 1000;
```

  - You would very quickly grow tired of typing variables names. And what happens if you change n to a million, nobody wants to manually create a million variables, let alone a thousand variables.
  - As you can see, one difficulty with lists is the naming of variables
  - There are various ways in which you represent list in C. The simplest is through an **array**

Arrays

Array

- An array is a contiguous block of memory that occupies
  $n \times size\_of\_data\_type$ bytes
  - Example:
    With $n = 3$ and a data type of char,
    the array would occupy
    $3 \times 1 = 3$ bytes
  - Example:
    With $n = 1000$ and a data type of `int`, the array would occupy
    $1000 \times 4 = 4000$ bytes
    or 4K in memory as each `int` occupies 4 bytes in memory
  - Example:
    If $n = 1000000$ and the data type was a `double`, the occupied space in memory would be
    $1000000 \times 8 = 8000000$ bytes
    or 8MB memory



Array

- Access array elements (variables) in C
  - Through a notation similar to subscripting in mathematics
  - Rather than subscripts which you can't represent using the ASCII character set, C uses the notation **`a[i]`** to indicate the *i*-th element in the array. *i* is called the **index** of the array
- Important distinction between mathematical and programming notation
  - The first element of the array is always `a[0]`, not `a[1]`. That is, **array indexing begins with zero in C**
  - So, for the sequence 1, 2, 3, 4, 5, …, 998, 999, 1000, the first element in C would be represented by

    `a[0] = 1`,

    and the last element by

    `a[999] = 1000`

Array

- Array declaration
  - Syntax: `type name[length]`
    - `type` is the data type of the array
    - `name` is the name you use to reference the array
    - `length` is the number of elements in the array, which is always one more than the last index of the array
  - Example, create an array to hold a 1000 integers

    ```
    int a[1000];
    ```
  - Another way to initialize array, which makes your code more maintainable, is to use a preprocessor constant

    ```
    const int SIZE = 1000;
    int a[SIZE];
    ```
  - Like other variables in C, declaring them does not initialize them

Array

- Access array elements (variables) in C
  - Through a notation similar to subscripting in mathematics
  - Rather than subscripts which you can't represent using the ASCII character set, C uses the notation **`a[i]`** to indicate the *i*-th element in the array. *i* is called the **index** of the array
- Array value assignment
  - Example

```
const int SIZE = 10;
int a[SIZE];

a [0] = 100; /* first element is set to 100 */
a [1] = 50; /* second element is set to 50 */
a [8] = 50;
a [9] = 100; /* last element is set to 100 */
```

Array

- Array initialization
  - If you would like to initialize every elements to the list to zero, you could do something like the following:

```
const int SIZE = 10;

int a[SIZE] = {0};
```

  - Another way you could initialize an array is a `for` loop. The `for` loop visits each element of the array, setting it to zero

```
const int SIZE = 10;

int a[SIZE];
int i = 0;

for (i = 0; i < SIZE ; i++)
        a[i] = 0;
```

Array

- Array initialization
  - If you wanted to set the array to the sequence 1, 2, …, 999, 1000, you would do this:

```
const int SIZE = 10;

int a[SIZE];
int i;

for (i = 0; i < SIZE ; i++)
        a[i] = i + 1;
```

  - Note how the `for` loop increments the index of the array. This allows the `for` loop to visit every element of the array and make an assignment. Or put another way, we say the `for` loop iterates over the array
  - Using the index in the `for()` statement and in the body of the `for` loop is a very very common way to process array elements

Array

- Array initialization
  - Another way to declare and initialize array is to explicitly define the array elements:

```
int a[] = {7, 5, 6, 7, 3, 4, 8, 9, 10, 11, 14};
```

  - Notice that there is no number between the brackets.
  - You do not need to put the size of the array if you declare and initialize on the same line.
- What is the length of the array in the previous example?
  - The compiler figures out the size for you and allocates space for 11 `int`s and makes the assignments.
  - This is convenient as you let the compiler figure out the size for you, but functions need to know the length of the array to work correctly.
  - → The compiler provides no magic in this case

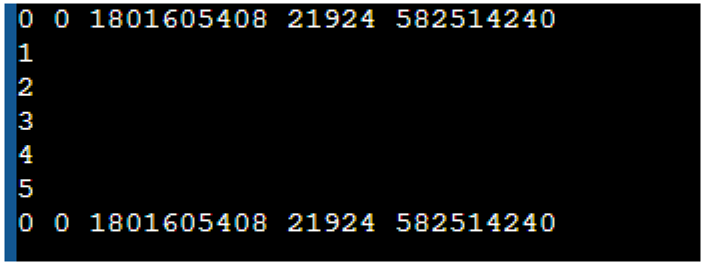
Foreach Statement in Arrays

- Using foreach statement to output elements in an array

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

for (auto i : a)
    cout << i << " ";
```

- Use the similar way to initialize the arrays

```
int a[5];

for (auto i : a)
    cout << i << " ";
cout << endl;

for (auto i : a)
    cin >> i;

for (auto i : a)
    cout << i << " ";
cout << endl;
```

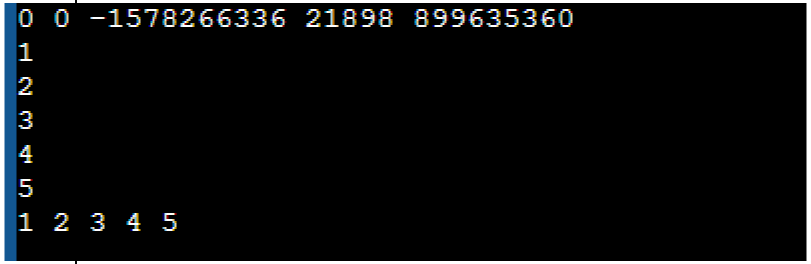


```
0 0 1801605408 21924 582514240
1
2
3
4
5
0 0 1801605408 21924 582514240
```

Foreach Statement in Arrays

- The correct way to write data in an array with foreach statement

```
int a[5];

for (auto i : a)
    cout << i << " ";
cout << endl;

for (auto& i : a)
    cin >> i;

for (auto i : a)
    cout << i << " ";
cout << endl;
```



```
0 0 -1578266336 21898 899635360
1
2
3
4
5
1 2 3 4 5
```

`auto` and `auto&`

- Choose `auto x` when you want to work with copies.
- Choose `auto& x` when you want to work with original items and may modify them.

- References:
  - https://stackoverflow.com/questions/29859796/c-auto-vs-auto
  - https://iamsorush.com/posts/auto-cpp/

Character Arrays/C Strings

Review - Characters

- Characters in C/C++
  - Declared as a `char` type.
  - Example:
    ```
    char myChar;
    ```
  - Assign a character to a char variable: use single quotes
  - Example:
    ```
    myChar = 'm';
    ```
  - Single quote has different usage than double quote

Characters

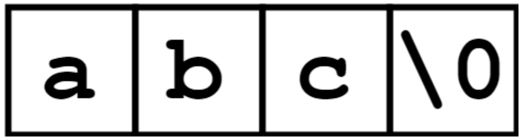
- Characters in C/C++
  - Declared as a `char` type.
  - Example:
    ```
    char myChar;
    ```
  - Assign a character to a char variable: use single quotes
  - Example:
    ```
    myChar = 'm';
    ```
  - Single quote has different usage than double quote
- Array of characters
  - Example: `char c[] = {'L', 'i', 'n', 'u', 'x'};`
  - Important: an array of character is **NOT** a string

String Literals

- Definition
  - A sequence of characters enclosed within double quotes
  - Example: `"Hello World!"`
  - Often appear as format strings in calls of `cout`
- Escape Sequence in String Literal
  - A sequence of characters is translated into another character that may be difficult or impossible to represent directly
  - `\` (backslash, escape character)
    - Is used to escape characters that otherwise have a special meaning
  - Example
    - `\n`: the new line character
    - `\t`: ASCII horizontal tab
    - `\"`: Double quote
    - `\\`: Backslash

String Literals

- How String Literals are Stored
  - C/C++ treats string literals as character arrays
  - The memory contains the characters in the string, plus one extra character - the null character - to mark the end of the string
  - Example: the string literal `"abc"`
    is stored as an array of four characters



  - `\0`: null character
- String Literals in C/C++
  - Since a string literal is stored as an array, the compiler treats it as a pointer of type `char *`
  - Example: `cout << "abc";`
    Pass the address of `"abc"` to `cout`

String Variables

- Definition
  - A sequence of characters enclosed within double quotes
  - Example: `"Hello World!"`
  - Often appear as format strings in calls of `cout`
- Declaration: using character arrays
  - Example: `char d[] = "hello";`
- Example

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

int main()
{
        /* Why does this fail? */

        char d[] = 'hello';
        cout << d << endl;
        return 0;
}
```

String Variables

- Example

```
#include <iostream>

using namespace std;

int main()
{
    char d[] = "hello";
    //this generates a compiler warning, which shouldn't be ignored
    d[0] = "x";
    cout << d << endl;
    return 0;
}
```

String Variables

- Example

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

int main() {
   char userStr[20] = "1234567890123456789";
   int i;

   cout << "Enter string (<20 chars): ";
   cin >> userStr;

   cout << endl << userStr << endl;

   for (i = 0; i < 20; ++i) {
      cout << userStr[i];
   }

   for (i = 0; i < 30; ++i) {
      cout << userStr[i];
   }

   return 0;
}
```

String Variables

- The correct way to find the length of a string:

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

unsigned int str_len(char s[])
{
	unsigned int i;
	for(i = 0; s[i] != '\0'; i++)
		;

	return i;
}

int main(void)
{
	char s[] = "hello, goodbye";
	unsigned int len = str_len(s);

	cout << "len of string is " << len << endl;

	return 0;
}
```

String Variables

- Declaration: using character arrays
  - Example: `char d[] = "hello";`
- Declaration: using character pointers
  - Example: `char *d = "hello";`
- Example

```
#include <iostream>

using namespace std;

int main()
{
    char *d = "hello";
    // cause some problems
    d[0] = 'x';
    cout << d << endl;
    return 0;
}
```

String Variables

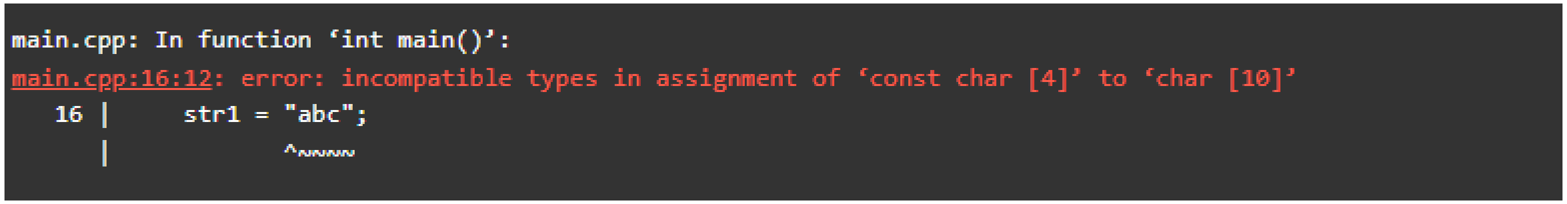
- Difference between character array and character pointers
  - Character Arrays
    - Declaration
      ```
      char c[] = "hello";
      ```
    - Is an array big enough to hold `hello` and `\0`
    - Can change individual characters of the array (mutable)
  - Character Pointers
    - Declaration
      ```
      char *s = "hello";
      ```
    - Is a pointer, initialized to point to a string constant or string literal
    - Cannot modify the string contents (immutable)
- More comparisons
  - https://www.geeksforgeeks.org/char-vs-stdstring-vs-char-c/

C Strings Library

C String

- Direct attempts to copy or compare strings will fail
  - String copy - Incorrect example 1

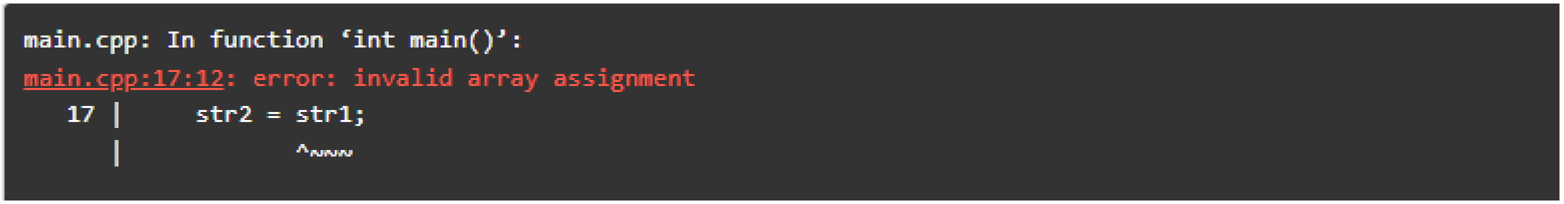
```
char str1[10];
str1 = "abc";
```



```
main.cpp: In function ‘int main()’:
main.cpp:16:12: error: incompatible types in assignment of ‘const char [4]’ to ‘char [10]’
   16 |     str1 = "abc";
      |            ^~~~~
```

  - String copy - Incorrect example 2

```
char str1[10] = "abc";
char str2[10];
str2 = str1;
```

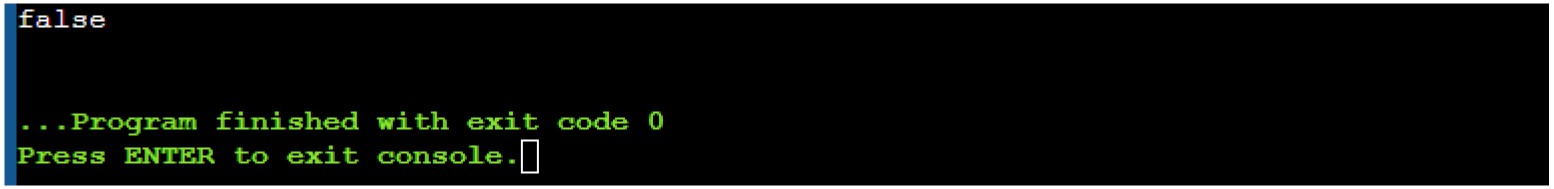


```
main.cpp: In function ‘int main()’:
main.cpp:17:12: error: invalid array assignment
   17 |     str2 = str1;
      |            ^~~~
```

C String

- Direct attempts to copy or compare strings will fail
  - String compare

```
char str1[10] = "abc";
char str2[10] = "abc";
if (str1 == str2)
    cout << "true" << endl;
else
    cout << "false" << endl;
```



```
false


...Program finished with exit code 0
Press ENTER to exit console.
```

  - The statement compares `str1` and `str2` as `pointers`; it doesn't compare the contents of the two arrays

<cstring> Library

strcpy()	`strcpy(destStr, sourceStr)`
	
	Copies sourceStr (up to and including null character) to destStr.
		`strcpy(targetText, userText); // Copies "UNICEF" + null char`
		`                               // to targetText`
		`strcpy(targetText, orgName);   // Error: "United Nations"`
		`                               // has > 10 chars`
		`targetText = orgName;          // Error: Strings can't be`
		`                               // copied this way`

strncpy()	`strncpy(destStr, sourceStr, numChars)` Copies up to numChars characters.	`strncpy(orgName, userText, 6); // orgName is "UNICEF Nations"`
strcat()	`strcat(destStr, sourceStr)` Copies sourceStr (up to and including null character) to *end* of destStr (starting at destStr's null character).	`strcat(orgName, userText); // orgName is "United NationsUNICEF"`

strncat()	`strncat(destStr, sourceStr, numChars)`
	
	Copies up to numChars characters to destStr's end, then appends null character.
		`strcpy(targetText, "abc");            // targetText is "abc"`
		`strncat(targetText, "123456789", 3); // targetText is "abc123"`

<cstring> Library

- Function strcpy and strncpy (String Copy) Example

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

int main()
{
	int const SIZE1 = 15, SIZE2 = 8;

	char x[] = "Hello World!";
	char y[SIZE1];
	char z[SIZE2];

	strcpy (y, x);
	strncpy (z, x, SIZE2 - 1);
	z[SIZE2 - 1] = '\0';

	cout << "x: " << x << endl;
	cout << "y: " << y << endl;
	cout << "z: " << z << endl;

	return 0;
}
```

<cstring> Library

- Function strcat and strncat (String Concatenation) Example

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

int main(void)
{
         char x[20] = "Hello ";
         cout << (sizeof(x)/sizeof(char)) << endl;
         char y[] = "World!";

         strcat (x, y);
         cout << "x: " << x << endl;

         strncat(x, y, 2);
         cout << "x: " << x <<endl;

         return 0;
}
```

`<cstring>` Library

strchr()	`strchr(sourceStr, searchChar)`
	
	Returns NULL if searchChar does not exist in sourceStr. (Else, returns address of first occurrence, discussed elsewhere).
	NULL is defined in the cstring library.
		`if (strchr(orgName, 'U') != NULL) { // 'U' exists in orgName?`
		`   ...  // 'U' exists in "United Nations", branch taken`
		`}`
		`if (strchr(orgName, 'u') != NULL) { // 'u' exists in orgName?`
		`   ...  // 'u' doesn't exist (case matters), branch not taken`
		`}`

strlen()	`size_t strlen(sourceStr)`
	
	Returns number of characters in sourceStr up to, but not including, first null character. size_t is integer type.
		`x = strlen(orgName);    // Assigns 14 to x`
		`x = strlen(userText);   // Assigns 6 to x`
		`x = strlen(targetText); // Error: targetText may lack null char`

strcmp()	`int strcmp(str1, str2)`
	
	Returns 0 if str1 and str2 are equal, non-zero if they differ.
		`if (strcmp(orgName, "United Nations") == 0) {`
		`   ... // Equal, branch taken`
		`}`
		`if (strcmp(orgName, userText) != 0) {`
		`   ... // Not equal, branch taken`
		`}`

<cstring> Library

- Function `strcmp` and `strncmp` (String Comparison) Example

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

int main(void)
{
         char s1[] = "Hello World!";
         char s2[] = "Hello World!";
         char s3[] = "Hello Tech";

         cout << "Compare s1 and s2: " << strcmp(s1, s2) << endl;
         cout << "Compare s1 and s3: " << strcmp(s1, s3) << endl;
         cout << "Compare s3 and s1: " << strcmp(s3, s1) << endl;
         cout << "Compare s1 and s3 with first 5 characters: "
                  << strncmp(s1, s3, 5) << endl;

         return 0;
}
```

String Library

Review - String Variables

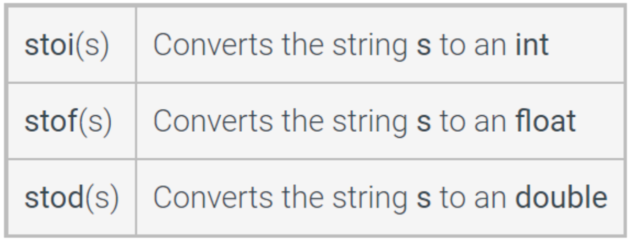
- Difference between character array and character pointers
  - Character Arrays
    - Declaration

      ```
      char c[] = "hello";
      ```
    - Is an array big enough to hold `hello` and `\0`
    - Can change individual characters of the array (mutable)
  - Character Pointers
    - Declaration

      ```
      char *s = "hello";
      ```
    - Is a pointer, initialized to point to a string constant or string literal
    - Cannot modify the string contents (immutable)
- More comparisons
  - https://www.geeksforgeeks.org/char-vs-stdstring-vs-char-c/

Conversion Functions

- Conversion Functions



stoi(s)	Converts the string s to an int
stof(s)	Converts the string s to an float
stod(s)	Converts the string s to an double

- Example

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

int main() {
    string s = "123", t = "4.5";
    int n = stoi(s);
    double x = stod(t);
    cout << n << " + " << x << " = " << n+x << endl;
}
```

The "s" Suffix

- Make a string literal to be interpreted as a std::string instead of a character array
  - #include <string> header
  - Place an s after the string literal

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

int main() {
    auto s1 = "Hello";
    cout << s1 << " is a " << typeid(s1).name() << endl;
    auto s2 = "Goodbye"s;
    cout << s2 << " is a " << typeid(s2).name() << endl;
}
```

Vector

Issues in Arrays

- Does not check whether subscripts fall outside the range of an array
- Can not be meaningfully compare with equality operators or relational operators
- Size of the array can not be passed to a general-purpose function (will be discussed later)
- One array can not be assigned to another with the assignment operators

`vector` Template Class

- A more robust type of array featuring many additional capabilities
  - Create a more powerful and less error-prone alternative to arrays
  - Use template notation (will be discussed later)
- Declaration
  - Have to include `<vector>` library:
    ```
    #include <vector>
    ```
  - Syntax:
    ```
    vector<dataType> vectorName(numElements);
    ```
  - Example:
    ```
    vector<int> itemCounts(3);
    ```
  - The angle brackets (or chevrons, `<>`): defines the type of each vector element.
  - The parentheses(`()`) following the vector name: specify the number of vector elements in the vector

`vector` Template Class

- Access Element
  - Using `.at()` method
    Examples:
    ```
    itemCounts.at(0) = 122;
    cout << itemCounts.at(1);
    oldestPeople.at(nthPerson - 1)
    ```
  - Using `[]` method
    Example: `itemCounts[1]`
- Initalization
  - Initialize all vector elements with a single value. Example:
    ```
    vector<int> myVector(3, -1);
    ```
    creates a vector named `myVector` with three elements, each with value `-1`
  - Initialize each vector element with different values: using braces `{}`. Example:
    ```
    vector<int> carSales = {5, 7, 11};
    ```
    creates a vector of three integer elements initialized with values `5`, `7`, `11`.

`vector` Template Class

- Getting the number of vector elements
  - Using `.size()` method
  - Example:
    `userVals.size()`
- Iterating through vectors using loops
  - Using for statement

```
for (i = 0; i < myVector.size(); ++i) {
   // Loop body accessing myVector.at(i)
}
```

  - Using foreach statement

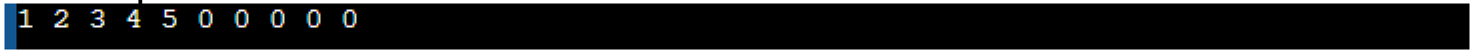
```
for (auto elem : myVector) {
       // Loopo body accessing elem
}
```

vector Operations

- Vector resize: using `.resize()` method
  - When the new size is larger, `.resize()` adds elements at the end; the new elements are default-initialized
    - Example:

```
vector<int> v = {1,2,3,4,5};
v.resize(10);
for (auto n: v)
    cout << n << " ";
cout << endl;
```

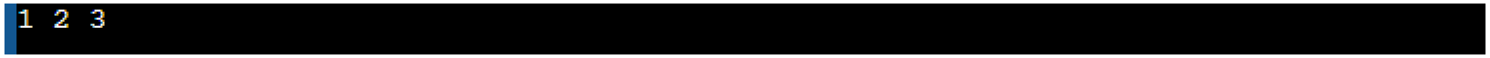
the output is:



```
1 2 3 4 5 0 0 0 0 0
```

  - When the new size is smaller, `.resize()` deletes elements from the end

```
vector<int> v = {1,2,3,4,5};
v.resize(3);
```

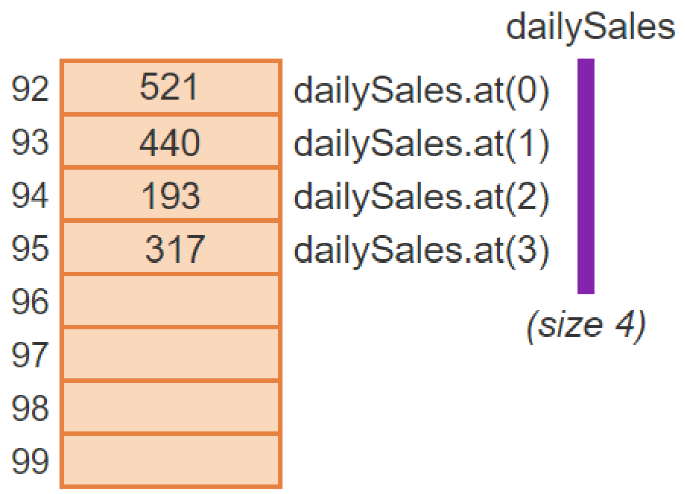


```
1 2 3
```

`vector` Operations

- Appending items to a vector: using `.push_back()` method
  - Example:

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

int main() {
   unsigned int i;
   vector<int> dailySales;

   cout << "Size before: " << dailySales.size();

   dailySales.push_back(521);
   dailySales.push_back(440);
   dailySales.push_back(193);
   dailySales.push_back(317);

   cout << ", after: " << dailySales.size() << endl;
   cout << "Contents:" << endl;
   for (i = 0; i < dailySales.size(); ++i) {
      cout << " " << dailySales.at(i) << endl;
   }

   return 0;
}
```



```
Size before: 0, after: 4
Contents:
521
440
193
317
```

Arrays vs. Vectors

Difference between `vectors` and Arrays

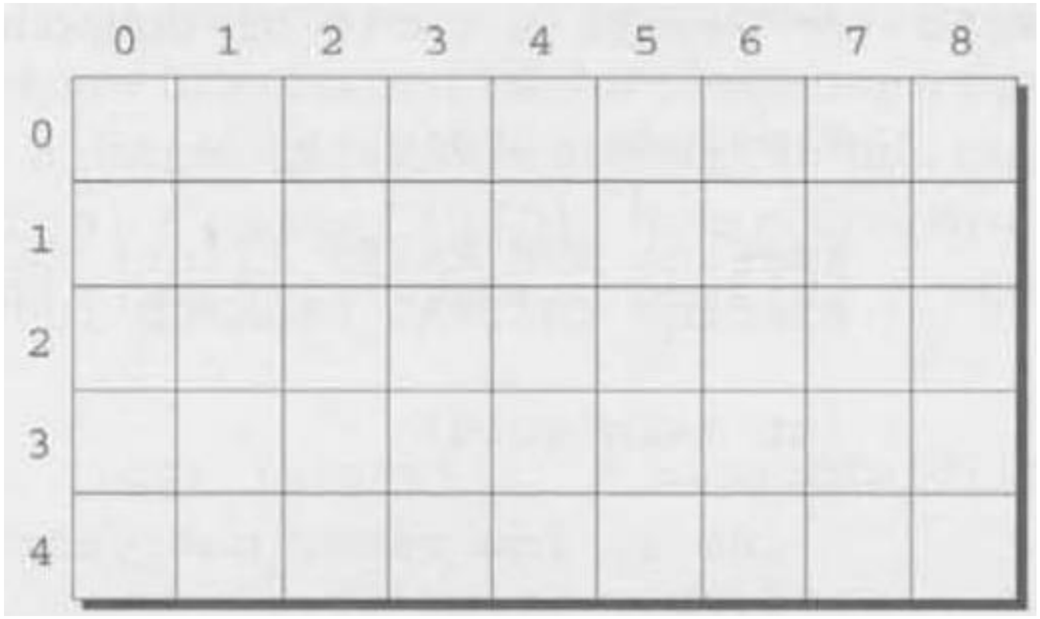
`vector`	Array
A sequential container to store elements and not index based	A fixed-size sequential collection of elements of the same type and it is index based
Dynamic in nature so, size increases with insertion of elements	Fixed size, once initialized can't be resized
Occupies more memory	Is memory efficient data structure
Takes more time in accessing elements	Access elements in constant time irrespective of their location as elements are arranged in a contiguous memory allocation

https://www.tutorialspoint.com/difference-between-std-vector-and-std-array-in-cplusplus

Multidimensional Arrays

Multidimensional Array

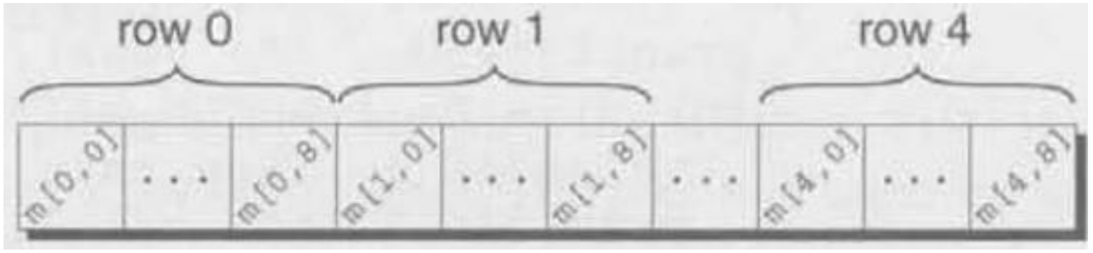
- Example: two-dimensional array
  - A matrix, in mathematical terminology
  - `int m[5][9];`
  - The array `m` has 5 rows and 9 columns. Both rows and columns are indexed from 0



  - To access the element of `m` in row `i`, column `j`, we must write `m[i][j]`

Multidimensional Array

- Row-major order
  - C/C++ stores array with row 0 first, then row 1, and so forth.
  - Example of the m array:



- Nested for loop

```
const int N = 10;

double indent[N][N];
int row, col;

for (row = 0; row < N; row++)
    for (col = 0; col < N; col++)
        if (row == col)
            ident[row][col] = 1.0;
        else
            ident[row][col] = 0.0;
```

Multidimensional Array

- Initializing a multidimensional array
  - Example:

```
int m[5][9] =   {{1, 1, 1, 1, 1, 0, 1, 1, 1},
                 {0, 1, 0, 1, 0, 1, 0, 1, 0},
                 {0, 1, 0, 1, 1, 0, 0, 1, 0},
                 {1, 1, 0, 1, 0, 0, 0, 1, 0},
                 {1, 1, 0, 1, 0, 0, 1, 1, 1}};
```

- Abbreviate initializer for multidimensional array:
  - If an initializer isn't long enough to fill a multidimensional array, the remaining elements are given the value 0
    - Example:

```
int m[5][9] =   {{1, 1, 1, 1, 1, 0, 1, 1, 1},
                 {0, 1, 0, 1, 0, 1, 0, 1, 0},
                 {0, 1, 0, 1, 1, 0, 0, 1, 0}};
```

The last two rows will contain zeros.

Multidimensional Array

- Abbreviate initializer for multidimensional array:
  - If an inner list isn't long enough to fill a row, the remaining elements in the row are initialized to 0
    - Example:

```
int m[5][9] =   {{1, 1, 1, 1, 1, 0, 1, 1, 1},
                  {0, 1, 0, 1, 0, 1, 0, 1, },
                  {0, 1, 0, 1, 1, 0, 0, 1, },
                  {1, 1, 0, 1, 0, 0, 0, 1, },
                  {1, 1, 0, 1, 0, 0, 1, 1, 1}};
```

The last elements in row 2, 3, and 4 are assigned to 0s.

  - We can omit the inner braces
    - Example:

```
int m[5][9] =   {1, 1, 1, 1, 1, 0, 1, 1, 1,
                  0, 1, 0, 1, 0, 1, 0, 1, 0,
                  0, 1, 0, 1, 1, 0, 0, 1, 0,
                  1, 1, 0, 1, 0, 0, 0, 1, 0,
                  1, 1, 0, 1, 0, 0, 1, 1, 1};
```

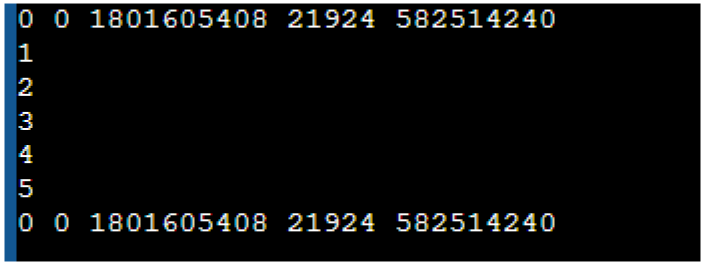
Review - Foreach Statement in Arrays

- Using foreach statement to output elements in an array

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

for (auto i : a)
    cout << i << " ";
```

- Use the similar way to initialize the arrays

```
int a[5];

for (auto i : a)
    cout << i << " ";
cout << endl;

for (auto i : a)
    cin >> i;

for (auto i : a)
    cout << i << " ";
cout << endl;
```

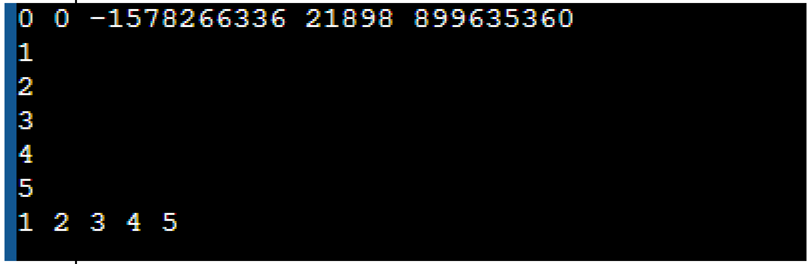


```
0 0 1801605408 21924 582514240
1
2
3
4
5
0 0 1801605408 21924 582514240
```

Review - Foreach Statement in Arrays

- The correct way to write data in an array with foreach statement

```
int a[5];

for (auto i : a)
    cout << i << " ";
cout << endl;

for (auto& i : a)
    cin >> i;

for (auto i : a)
    cout << i << " ";
cout << endl;
```



```
0 0 -1578266336 21898 899635360
1
2
3
4
5
1 2 3 4 5
```

Review - `auto` and `auto&`

- Choose `auto x` when you want to work with copies.
- Choose `auto& x` when you want to work with original items and may modify them.

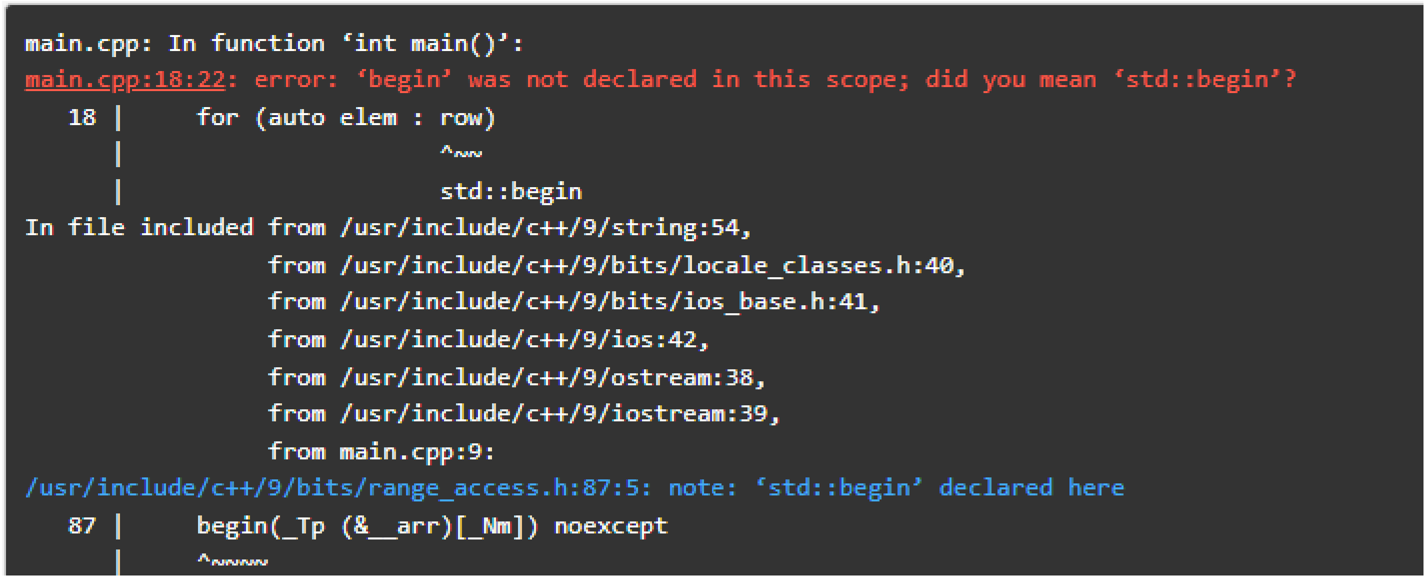
- References:
  - https://stackoverflow.com/questions/29859796/c-auto-vs-auto
  - https://iamsorush.com/posts/auto-cpp/

Foreach Statement in Multi-Dimensional Arrays

- Incorrect Statement for reading elements

```
int a[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

for (auto row : a) {
    for (auto elem : row)
        cout << elem << " ";
    cout << endl;
}
```

row: the copy of each row in array a



```
main.cpp: In function ‘int main()’:
main.cpp:18:22: error: ‘begin’ was not declared in this scope; did you mean ‘std::begin’?
   18 |     for (auto elem : row)
      |                      ^~~
      |                      std::begin
In file included from /usr/include/c++/9/string:54,
                 from /usr/include/c++/9/bits/locale_classes.h:40,
                 from /usr/include/c++/9/bits/ios_base.h:41,
                 from /usr/include/c++/9/ios:42,
                 from /usr/include/c++/9/ostream:38,
                 from /usr/include/c++/9/iostream:39,
                 from main.cpp:9:
/usr/include/c++/9/bits/range_access.h:87:5: note: ‘std::begin’ declared here
   87 |     begin(_Tp (&__arr)[_Nm]) noexcept
      |     ^~~~~
```

Foreach Statement in Multi-Dimensional Arrays

- Correct Statement for reading elements

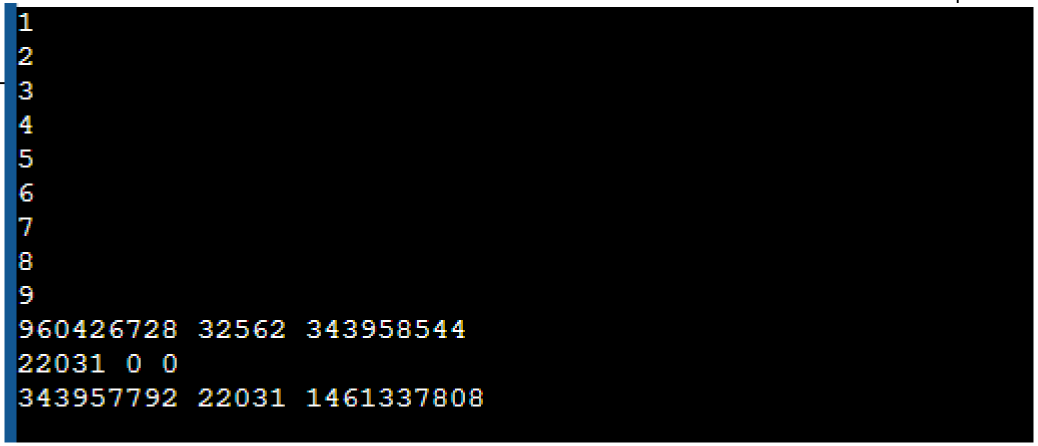
```
int a[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

for (auto& row : a) {
    for (auto elem : row)
        cout << elem << " ";
    cout << endl;
}
```

  - C++ does not like to copy array.
  - `row` in this sample code is the reference of each row in the array `a`

Foreach Statement in Multi-Dimensional Arrays

- Incorrect Statement for reading elements

```
int a[3][3];

for (auto& row : a)
    for (auto elem : row)
        cin >> elem;

for (auto& row : a) {
    for (auto elem : row)
        cout << elem << " ";
    cout << endl;
}
```



```
1
2
3
4
5
6
7
8
9
960426728 32562 343958544
22031 0 0
343957792 22031 1461337808
```

Foreach Statement in Multi-Dimensional Arrays

- Correct Statement for reading elements

```
int a[3][3];

for (auto& row : a)
    for (auto& elem : row)
        cin >> elem;

for (auto& row : a) {
    for (auto elem : row)
        cout << elem << " ";
    cout << endl;
}
```

  - Work with original items in the array elements and may modify them