

# Arrays and String

- Using array
- Initialized arrays of different types
- Null-terminated string
- character array — string
- Multidimensional arrays
- Generate a string by character array
- String objects

# Arrays

- An array of temperatures
  - `double temperatures[366];`
- Index
- Offset – distance between the first element to current element
- Program 6.1

# Initializing An Array

```
const int max_heights = 10;    //array size  
int height[max_heights];  
int samples[5] = {2, 3, 4, 6, 11};
```

- Program 6.2

# Initialized All Elements

- Initialize all elements to zero
  - `double junk[20] = {0};`
  - `double junk[20]={};`
- Declare array size by initial values
  - `int values[ ] = {2, 3, 4};`
  - `int values[3] = {2, 3, 4};`
- Using `sizeof()` to find the size of an array,  
Program 6.3

# C-style String and char Array

- C-style string ends up with a **null character** `'\0'`
  - 10 elements array end up with two `'\0'`
    - `char name[10] = "Mae West";`
  - 9 elements array end up with one `'\0'`
    - `char name[ ] = "Mae West";`
  - 6 elements array
    - `Char vowels[ ] = "aeiou";`
- An array of characters with
  - `char vowels[5] = { 'a', 'e', 'i', 'o', 'u' }; //five elements`
  - `char vowels[ ] = "aeiou"; // six elements`
- Program 6.4

## An array of type char

```
const int maxlength = 100;  
char text[maxlength] = {0};  
// input ended at ' ' or '\n', regarded as a delimiter  
cin >> text;  
// input ended at '\n'  
cin.getline(text, maxlength);  
// input many lines ended at '!'  
cin.getline(text, maxlength, '!');
```

# Multidimensional Array

```
double carrots[3][4];
```

This row is `carrots[0]`

<code>carrots[0][0]</code>	<code>carrots[0][1]</code>	<code>carrots[0][2]</code>	<code>carrots[0][3]</code>
----------------------------	----------------------------	----------------------------	----------------------------

This row is `carrots[1]`

<code>carrots[1][0]</code>	<code>carrots[1][1]</code>	<code>carrots[1][2]</code>	<code>carrots[1][3]</code>
----------------------------	----------------------------	----------------------------	----------------------------

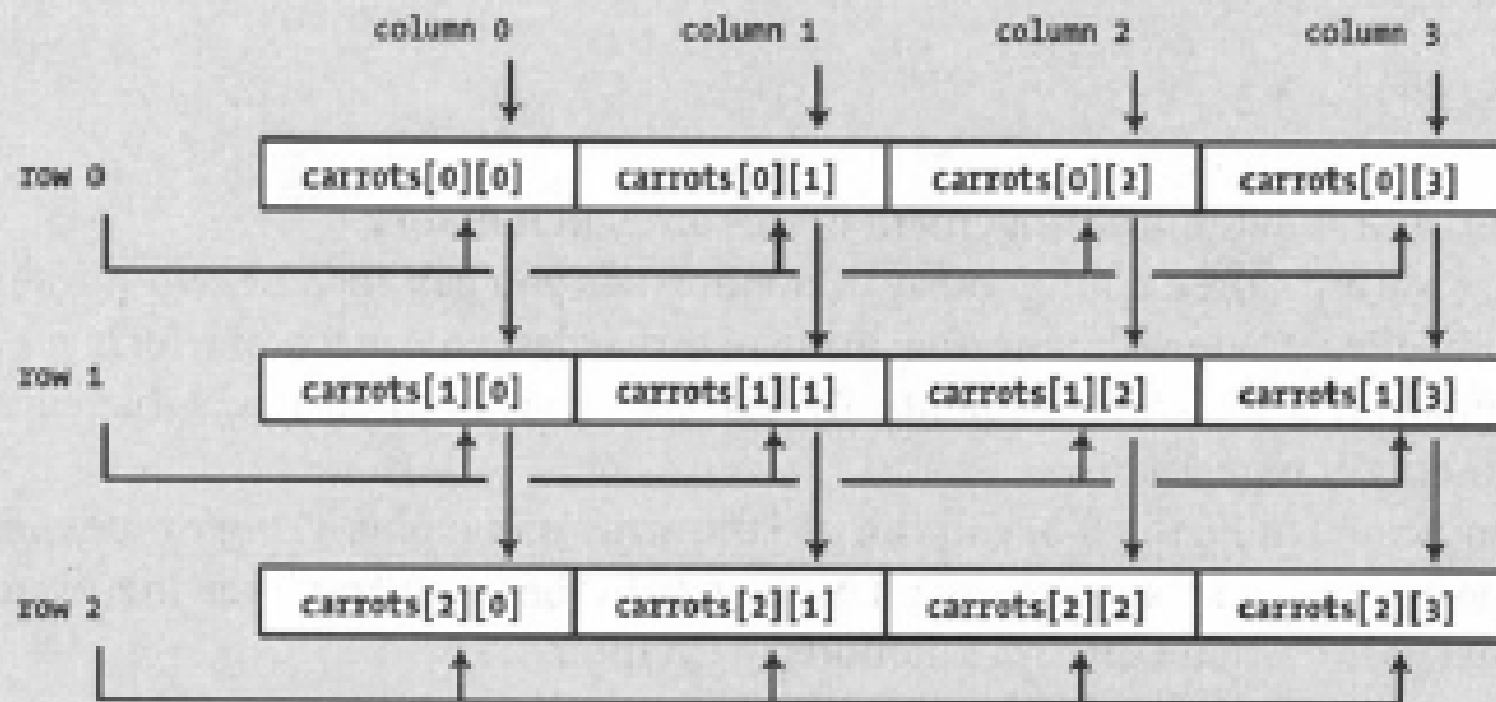
This row is `carrots[2]`

<code>carrots[2][0]</code>	<code>carrots[2][1]</code>	<code>carrots[2][2]</code>	<code>carrots[2][3]</code>
----------------------------	----------------------------	----------------------------	----------------------------

You can refer to the whole array as `carrots`.

# Rows and Columns

```
double carrots[3][4];
```





# Multidimensional Array

```
double carrots[3][4];  
for (int i=0; i<3; i++)  
{  
    for (int j=0; j<4; j++)  
        cout << setw(12) << carrots[i][j];  
    cout << endl;  
}
```

# Multidimensional Array

```
double carrots[3][4];  
for (int i=0; i<sizeof carrots/sizeof carrots[0]; i++)  
{  
    for (int j=0; j<sizeof carrots[0]/sizeof carrots[0][0];  
        j++)  
        cout << setw(12) << carrots[i][j];  
    cout << endl;  
}
```

# Multidimensional Array

```
const int nrows = 3;
const int ncols = 4;
double carrots[nrows][ncols];
for (int i=0; i<nrows; i++)
{
    for (int j=0; j<ncols; j++)
        cout << setw(12) << carrots[i][j];
    cout << endl;
}
```

# Initializing Multidimensional Arrays

```
double carrots[3][4] = { {2.5, 3.4, 4.6, 6.7},  
                        {3.6, 5.3, 4.3, 2.9},  {2.3, 4.4, 1.0, 2.0} };
```

- Zeros for the non-initialized elements

```
double carrots[3][4] = {{2.5, 3.4}, {3.6},  
                        {2.3, 4.4, 1.0}};
```

```
double carrots[3][4] = {0};
```

```
double carrots[3][4] = {1.1, 1.2, 1.3, 1.4,  
                        1.5, 1.6};
```



# String Array

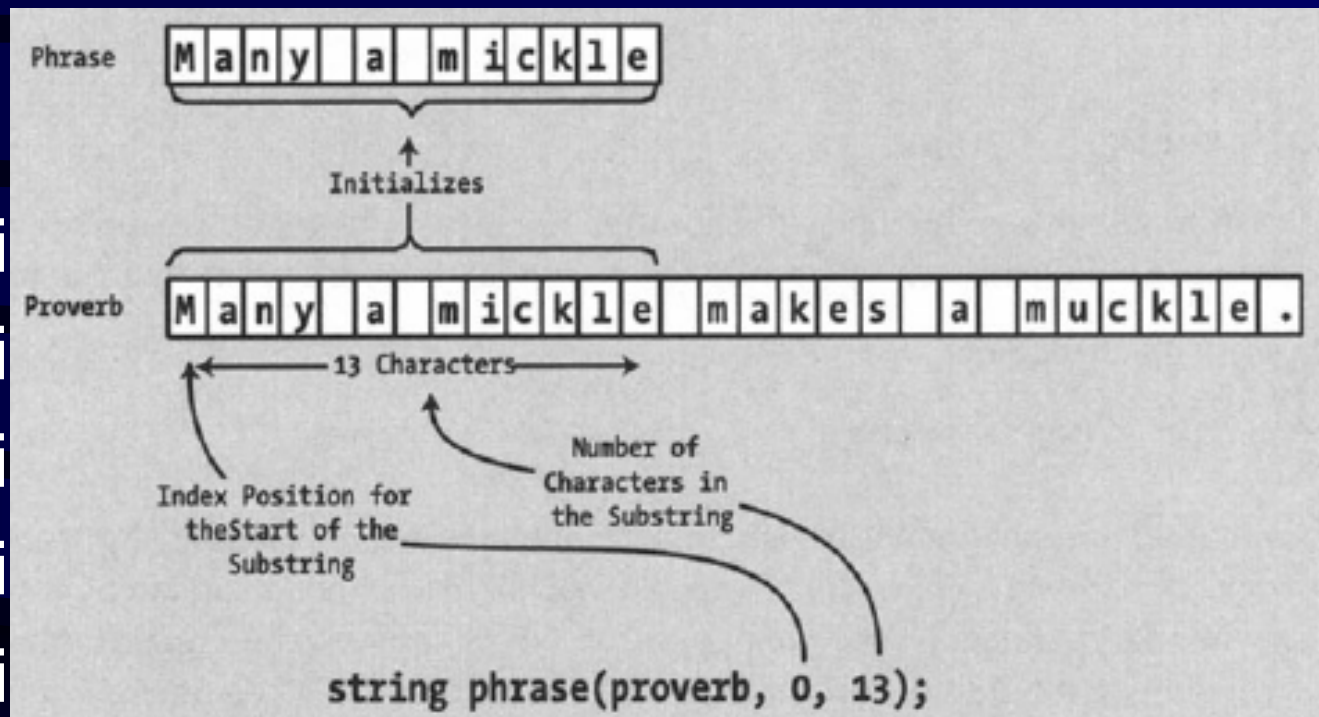
```
char start[6][80] = {  
    "Robert Redford",  
    "Hopalong Cssidly",  
    "Lassie",  
    "Slim Pickens",  
    "Boris Karloff",  
    "Oliver Hardy" };
```

- Program 6.5

# string Class

- Declaring an empty string object
  - `string myString;`
- Initialize a string object
  - `string proverb = "Many a mickle makes a muckle";`
  - `string proverb("Many a mickle makes a muckle");`

# string Class



zzzzzz";  
= "Z";

string phrase(proverb, 0, 13); // see p.201



# string Class

```
string adjective = "hornswoggling";
```

```
string word = "rubbish";
```

```
word = adjective;
```

```
adjective = "twotiming";
```

# String Concatenation

```
string description = adjective + " " +  
    "whippersnapper";
```

- Program 6.6

# Read a Line

- Read a line from keyboard and store in text  
    string text;  
    getline(cin, text);
- Read a line from keyboard and store in text until '#'  
    getline(cin, text, '#');
- Program 6.7

# Some Overloaded Functions in `string.h`

- `substr()`
- `compare()`
- `find()`, `rfind()`
- `find_first_of()`, `find_not_first_of()`
- `find_last_of()`, `find_not_last_of()`
- `insert()`, `replace()`, `erase()`

# Access Substring

```
string phrase = "The higher the fewer";  
string word = phrase.substr(4, 6);    //  
    "higher"  
word = phrase.substr(4, 100); // till the  
    end  
word = phrase.substr(4); // till the end
```

# String Comparison

- String comparing operations are associated to ASCII sequence
- Program 6.8

# The compare() Function

- `object_name.compare(other_object)`

```
for $ if(word1.compare(2, 4, "jack") == 0)
: $    std::cout << "Equal" << std::endl;    ;
it(word1.compare(2, 4, word2, 8, 4) == 0)
' if(word1.compare(2, 4, "jacket", 4) == 0) )
    std::cout << "Equal" << std::endl;
```

```
if(word1.substr(2,4) == word2.substr(8,4))
    std::cout << "Equal" << std::endl;
```

# The find() Function

```
string sentence = "Manners maketh man";
string word = "an";
int count = 0;                                     // Count of occurrences
size_t position = 0;                               // Stores a string index position
for(size_t i = 0 ; i<sentence.length()-word.length() ; ) {
    position = sentence.find(word, i);
    std::cout << sentence.find("akat", 1, 2) << std::endl; // Outputs 9
    // break;
    std::cout << sentence.find("akat", 1, 3) << std::endl; // Outputs string::npos
    std::cout << sentence.find("akat", 10, 2) << std::endl; // Outputs string::npos
    i = position+1;
}
std::cout << "\"" << word << "\" occurs in \"" << sentence
    << "\" " << count << " times.";
```

- Programs 6.9



# The find\_first\_of() Functions

```
1 string sentence = "Manners maketh man";
  string word = "an";
  std::cout << sentence.rfind(word) << std::endl;    // Outputs 16
  std::cout << sentence.rfind("man") << std::endl;    // Outputs 15
  std::cout << sentence.rfind('e') << std::endl;    // Outputs 11
  ...
  << endl;
  word_count++;                                     // Increase the count

  // Find the first character of the next word
  start = text.find_first_not_of(separators, end + 1);
}

  << std::endl;
string separators = " ,.\\\"";
std::cout << text.find_first_of(separators)           // Outputs 5
  << std::endl;
```

# The insert(), replace() and erase() Functions

- **Insert** a sub-string into the specific position in a string object
- **Replace** all the sub-strings in a string object
- **Erase** a sub-string in a string, or removing characters from a string
- Program 6.11

# Arrays of Type string

```
string words[] = {"this", "that", "the other"};  
string words[10] = {"this", "that", "the other"};  
words[2][6] = 't';  
cout << words[2]; // "the otter"
```