

STRINGS AND ARRAYS

Today

- This lecture will finish what we need to cover on strings:
 - Functions that have strings as parameters.
- We will also talk more about arrays:
 - Functions that take arrays as parameters
 - Two-dimensional arrays
 - Arrays of strings
- Finally we will talk about input and output of characters.
- Most of the code used in this lecture appears in the file `strings-and-arrays.cpp` which you can download from the class website.

Functions that have string parameters

- We have plenty of experience now writing functions that have, for example, integer parameters.
- For example:

```
bool isItPositive(int number)
{
    if(number >= 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

- This takes one parameter, an integer, and returns true if the integer is positive, false if the integer is negative.
- What if we want to pass a string to a function?
- Well, since `string` is a datatype, we can just use `string` as the datatype of a parameter.
- For example:

```
int giveCombinedLength(string s1, string s2)
{
    return s1.length() + s2.length();
}
```

takes two strings as arguments, and returns an integer that is the sum of the lengths of the two strings.

- We can also have a string as a return type.
- This (rather silly) function:

```
string oddOrEven(int number)
{
    if(number % 2 == 0)
    {
        return "even";
    }
    else
    {
        return "odd";
    }
}
```

takes an integer as its argument and returns the string even if the number is even, and the string odd if the number is odd.

- As for ints, chars and doubles, we can pass strings as reference parameters.

- The function prototype:

```
void noChange(string s)
```

is for a function that does not have a string reference parameter, while

```
void change(string &s)
```

is for a function that does have a string reference parameter.

- The program `more-strings.cpp`, on the course website, illustrates the use of reference parameters with strings.

An array of strings

- Just as we can declare an array of integers, we can declare an array of strings.
- To extend our DNA example, we can declare an array that represents three genes:

```
string genes[3] = {"tatagg",
                  "gagattc", "cgcgttat"}
```

- A member of this array is then a string, and we can call do everything to it that we can do to a string.
- For example:

```
genes[1].length();
```

will return 7, the length of `gagattc`.

- Because we can treat each string in `genes` as an array, we can pick out an individual character from one of the members of `genes`.

- Thus:

```
genes[2][1];
```

will return a `g`.

Two-dimensional arrays

- The arrays we have seen so far have allowed us to represent lists of things.
- We can also represent lists of lists.
- The declaration

```
int grid[2][3];
```

declares an array that has two three element arrays of integers.

- We call such an array *two dimensional*.
- As with the arrays we have seen before, we can combine declaring and initialising these arrays:

```
int grid[2][3] = {{1, 1, 1}, {2, 2, 2}};
```

- When we handle arrays with one dimension, we typically use a for loop.
- When we handle arrays with two dimensions, we typically use for loops that are *nested*.
- For example:

```
for(i = 0; i < 2; i++)
{
    for(j = 0; j < 3; j++)
    {
        cout << grid[i][j] << endl;
    }
}
```

- `grid[i][j]`, of course, identifies a single integer.

Sending arrays to functions

- We call a function on an array as follows:

```
void printArray(int a[])
```

- This is a function with an argument that is a one dimensional array of integers.
- Note that we don't need to say how big the array is.
- If we have a two dimensional array as a function parameter, we have to say how big the second dimension is:

```
void printGrid(int g[][3])
```

- Even for a one dimensional array, it is often *helpful* to say how big the array is.
- It can make our functions more useful, for example.

```
void printArray(int a[], int size)
{
    int i;

    for(i = 0; i < size; ++i){
        cout << a[i] << " ";
    }
}
```

can be called on arrays of different sizes, see `strings-and-arrays.cpp`.

- Without the parameter `size`, we'd have to say what the upper limit of the value of `i` was.
- That would make the function less flexible.

- Arrays are *always* sent to functions as reference parameters.
- To see an example of this, see `strings-and-arrays.cpp`

Character input

- (We didn't look at this in class, but it is still useful to know.)
- When we have considered reading strings in from files, and outputting strings, we have always thought about the whole string.
- We can also do it character by character.
- Once we have declared:

```
char c;  
ifstream myfile;  
myfile.open("inputfile.txt");
```

we can use:

```
c = myfile.get();
```

to read in a character from `inputfile.txt`.

- Similarly, following:

```
ostream myOtherFile;  
myOtherFile.open("outputfile.txt");
```

we can use:

```
myOtherFile.put(c);
```

to send a character to `outputfile.txt`.

Summary

- This lecture has finished our discussion of strings and arrays.
- We looked at functions that operate on strings.
- We looked at arrays of strings.
- We looked at multi-dimensional arrays.
- We looked at functions that operate on arrays.
- Finally, these slides looked at character input and output.