Lecture 7
Strings and Vectors

TIC1001 Introduction to Computing and Programming I

New Control Flow Instruction

switch-case

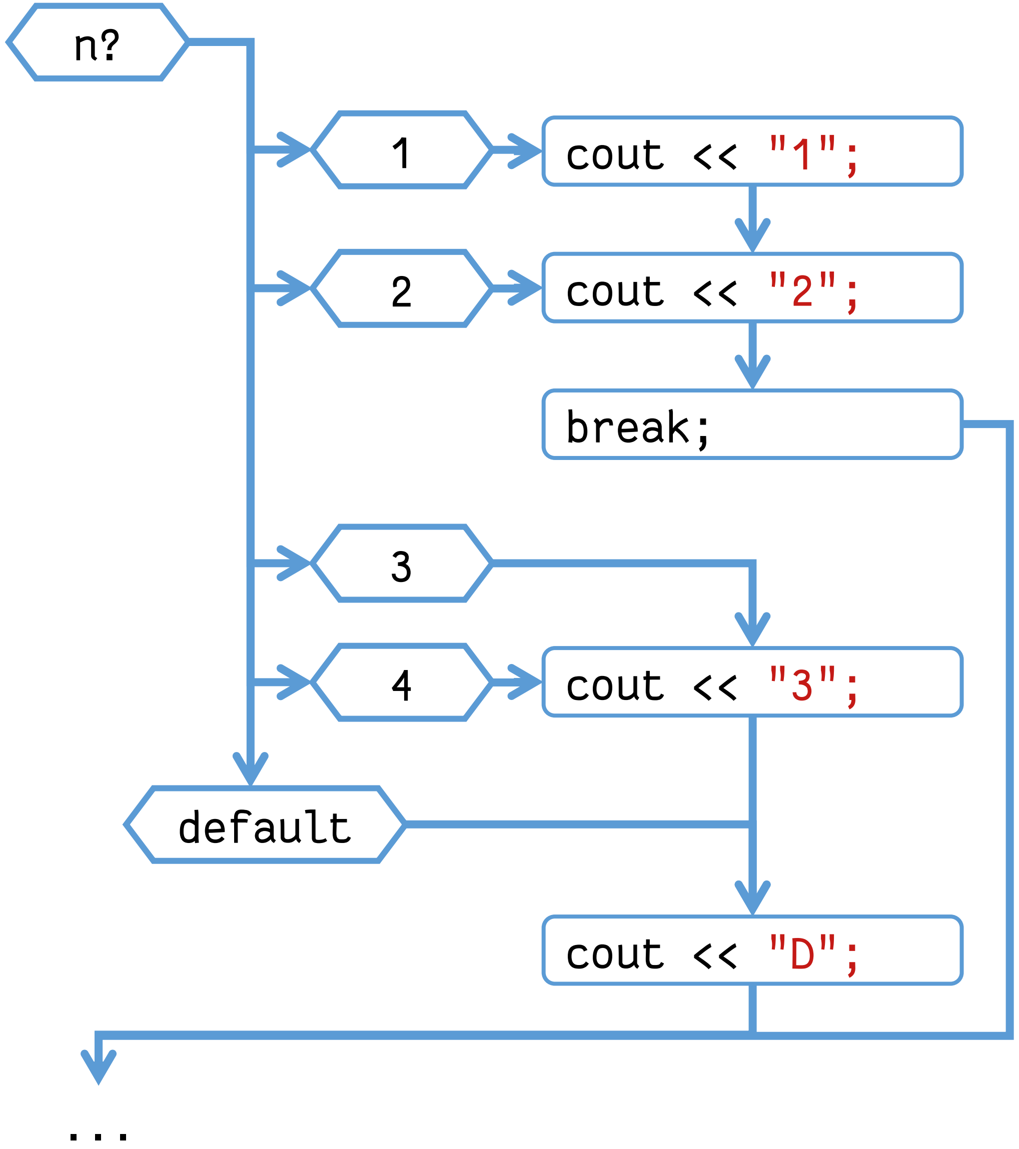
- Like multiple if-else-if

```
switch (<cond>) {
    case <c1>: statements
    case <c2>: statements
    default: statements
}
```

- <cond> must evaluate to an integer (or enum) type
- program will jump to the case that matches the <cond>
- or enter `default`, if present
- continue until end of switch, or `break` is reached

Switch-case Example

```
void f(int n) {
    switch(n) {
    case 1:
        cout << "1";
    case 2:
        cout << "2";
        break;
    case 3:
    case 4:
        cout << "3";
    default:
        cout << "D";
  }
}
```



Recap: C String

Just a special case of character array

- Uses string terminator
- Uses functions defined in <string.h>

Limitations

- Fixed size upon declaration
- Null terminator is error prone
- Require string functions to use
  - `strcpy` to assign strings
  - `strcmp` to compare strings
  - `strlen` to get length.
  - `strcat` to concatentate

C++ std::string

A first foray into OOP

std::string

A C++ class to manipulate strings

- Not a primitive type, unlike C-String
- Internal mechanisms are abstracted away
- Provides higher level functionality

Using std::string

String is not a built-in primitive, requires directive

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

Declare with

```
string name;
```

- Unlike primitives, strings will be initialized to empty string

Strings can be assigned and reassigned anytime

```
string name = "Mary";
name = "Mary Poppins";
string new_name = name;
```

Comparing between std::string

Strings support common arithmetic comparators

```
string one = "one";
string two = "two";
string three = "three";

cout << (one < two) << endl;
cout << (two < three) << endl;
```

Compared using lexicographical ordering

– a.k.a ASCII order

Comparing std::string

Why doesn’t this work?

```
cout << ("one" < "two") << endl;
cout << ("two" < "three") << endl;
```

Concatenating std::string

Simply by using the + operator

```
string first = "Mary";
string last = "Poppins";
string name = first + " " + last;
```

+= operator can also be used

```
name += " Y'all!";
```

Accessing the characters

Similar to an array, strings can be indexed

```
char initial = first[0];
```

- returns a char

Characters of strings can be modified

```
name[0] = 'L';
cout << name << endl;
```

C++ String Example

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

int main() {
  string str1;
  string str2("xyz");

 str1 = "abc";
  cout << "S1 = " << str1 << endl;
  cout << "S2 = " << str2 << endl;
  cout << "S1 + S2 = " << str1 + str2 << endl;
  cout << "S2 + S1 = " << str2 + str1 << endl;

  if (str1 > str2)
     cout << "S1 > S2" << endl;
  else
     cout << "S1 <= S2" << endl;
}
```

str1 is an empty string

str2 initialized with "xyz"

"=" can be used to assign a string

Output:

```
S1 = abc
S2 = xyz
S1 + S2 = abcxyz
S2 + S1 = xyzabc
S1 <= S2
```

C++ String Example

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
  string str1("abcd");
  string str2("efgh");
  string str3;

  str3 = str1 + str2;

  cout << str3 << endl;
  cout << str3.size() << endl;
  cout << str3[4] << endl;
  cout << str3.at(4) << endl;
}
```

"Addition" returns a newly concatenated string

Output:

```
abcdefgh
8
e
e
cdefg
```

Length of std::string

Using member functions (or methods)

```
name.length();
name.size();
```

- Both are synonyms

Note the difference between regular functions for C-Strings

```
strlen(c_string);
```

Methods “belong” to an object

```
s_string.length();
```

More details in TIC1002

When to use C-String or std::string?

With C++, it is typically easier to just use std::string

- Cannot use with `printf`, or other formatting functions
- Convert to C-String first using this method

```
string.c_str()
```

Example:

```
string name = "John";
char *cname = name.c_str();
printf("%s is here.", cname);
```

C-String	vs	std::string
`char *s = "This is a string";`		`string s = "This is a string";`

`char *t;`
`strcpy(t, s);`
		`string t;`
		`t = s;`

`int l = strlen(s);`		`int l = s.length();`
`if (strcmp(s, t) < 0) …`		`if (s < t) …`
`strcat(t, s);`		`t += s;`

std::vector

Arrays on steroids

STL Vector

http://en.cppreference.com/w/cpp/container/vector

Header file

```
#include <vector>
```

Stores contiguous elements as an array

- i.e. OO implementation of array

Advantages

- Fast insertion and removal at end of vector
- Dynamic sizing
- Automatic memory management
- One of the STL container classes (more next semester)

Declaring a vector

Just like arrays, vectors have to be homogenous

- all elements belong to the same type
- cannot change type after declaration

Declaring vectors

```
vector<type> my_vector;
```

- Examples

```
vector<int> v_int;
vector<string> strings;
```

STL Vector: Common Methods

Initializing vectors

```
vector<int> my_vector = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

- this only works from C11 onwards
- GCC 6.1 compiles with C14 standard as default

Size/Length of vector

```
my_vector.size()
```

- Note the ( ) at the end
- It is a function call
- What happens when you accidentally omit the ( ) ? Try it.

STL Vector: Common Methods

Adding elements to a vector

```
my_vector.push_back(10);
```

- Adds new element to the back of the vector

Inserting elements

```
my_vector.insert(my_vector.begin()+5, 22);
```

- Inserts 22 into the index 5 of the vector
- All elements get “pushed” down

```
my_vector.insert(my_vector.end()-5, 22);
```

- Inserts 22 into the index 5 from the back of the vector

STL Vector: Common Methods

Erasing elements

```
my_vector.pop_back();
```

- Deletes the last element

```
my_vector.erase(my_vector.begin()+5);
```

- Erases the element at index 5
- All following elements gets “pushed” up

```
my_vector.erase(my_vector.end()-5);
```

STL Vector: Common Methods

Accessing the elements

```
int first = my_vector[0];  // first element
int second = my_vector.at(1);
int last = my_vector.[my_vector.size()-1];  // last element

first = my_vector.front();
last = my_vector.back();
```

Modifying the elements

```
my_vector[0] += 10;
my_vector.at(1) += 10;
my_vector.front() = 5;
```

STL Vector: Common Methods

Iterating through the vector

```
for (int i = 0; i < my_vector.size(); i++)
    my_vector[i] * my_vector[i];
```

Displaying a vector

```
cout << my_vector << endl;  // error!
```

- Cannot just use cout
- Have to manually iterate through the elements

```
for (int i = 0; i < my_vector.size(); i++)
    cout << my_vector[i] << ",";
```

STL Vector: Common Methods

vector<T> v	Construct a vector v to store elements of type T
size()	returns the number of items
empty()	returns true if the vector has no elements
clear()	removes all elements
at(n) or [n]	returns an element at position n
front()	returns a reference to the first element
back()	returns a reference to the last element
push_back(e)	add element e to the end
insert(pos, e)	add element e in given position iterator
pop_back()	deletes the last element
erase(pos)	deletes the element in the given position iterator
begin()	returns an iterator to the front
end()	returns an iterator to the back

Strings are vectors too

Well not exactly, But in some ways…

They have the same functions/methods

- front
- back
- insert
- erase
- push_back
- pop_back
- begin
- end

Using strings and vectors in functions

Value Semantics

The assignment operator copies the value over

```
int i = 0;
int j = i;
i = 1;
cout << i << j << endl;
```

Strings and Vectors

```
string s = "Hello World!";
string t = s;
string s[0] = "B";
cout << s << t << endl;
```

```
vector<int> v = {1, 2, 3, 4};
vector<int> u = v;
v[0] = 100;
cout << v[0] << u[0] << endl;
```

“Reference” Semantics

Arrays are actually Pointers

```
int a[10] = {1, 2, 3, 4, 5};
int b[] = a;
a[0] = 100;
cout << a[0] << b[0] << end;
```

Passing strings and vectors

Passing into function is pass-by-value

- A new copy is made in the function
- Elements are copied over
- Changes to string/vector in function does not affect input

What if we want the function to modify the string/vector

- Use pass-by-reference

```
void capitalize(string &input);
```