Data Structures, Algorithms and Applications in C++

Textbook By: Sartaj Sahni

What is Data?

- Representation of information
- Can be a single number, array of numbers or strings, 2D etc
- Data can be structured (labelled) or unstructured

What The Course Is About

- Data structures is concerned with the representation and manipulation of data.
- All programs manipulate data.
- Data manipulation requires an algorithm.

Definition: A data structure is a particular way of organizing data in a computer so that it can be used effectively.

C++ is the object oriented programming language used to write algorithms that can organize data.

We study Data Structures and Algorithms.

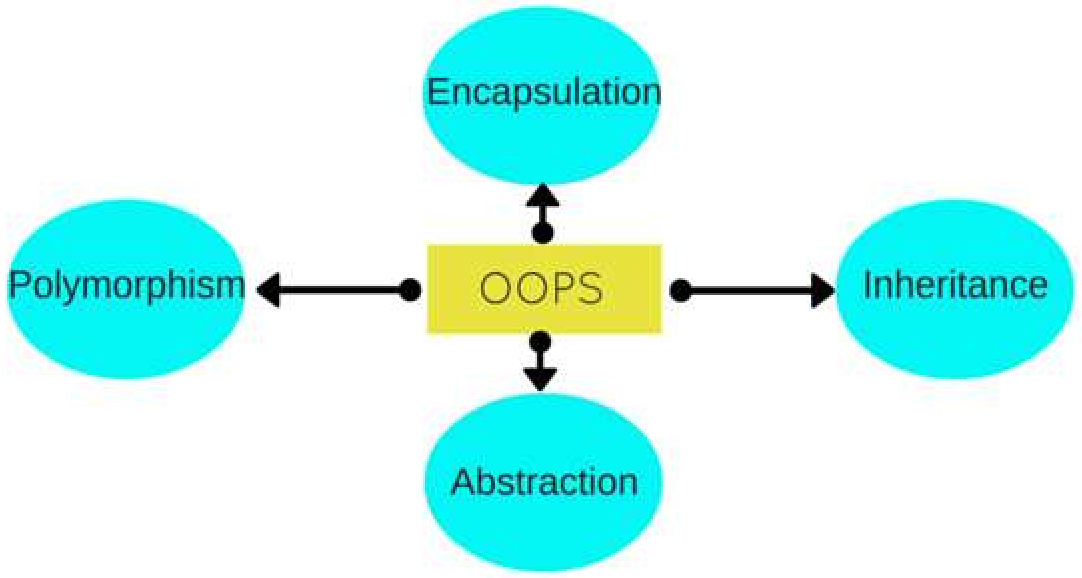
Data structures are used in computing to make it easy to locate and retrieve information.

Data Structures (Linear / Non Linear)	*Algorithms*
Lists, Stacks, Queues	Insert
Heaps	Delete
Binary Search Trees	Find
AVL Trees	Merge
Hash Tables	Shortest Paths
Graphs	Union
Disjoint Sets	

Revisiting C++ Concepts : Chapter 1

- Functions
- Template Parameters
- Function or Operator Overloading
- Exception Handling
- Dynamic Memory Allocation : new , delete
- Classes, Constructors, Destructors
- Testing : Black Box and White Box testing
- Debugging

Object Oriented programming is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation



- Class in C++ is defined as user defined data which declares & defines what data variables the object will have and what operations can be performed on the class's object.

- Objects are the basic unit of OOP. They are instances of class, which have data members and uses various member functions to perform tasks.

- Abstraction refers to showing only the essential features of the application and hiding the details.

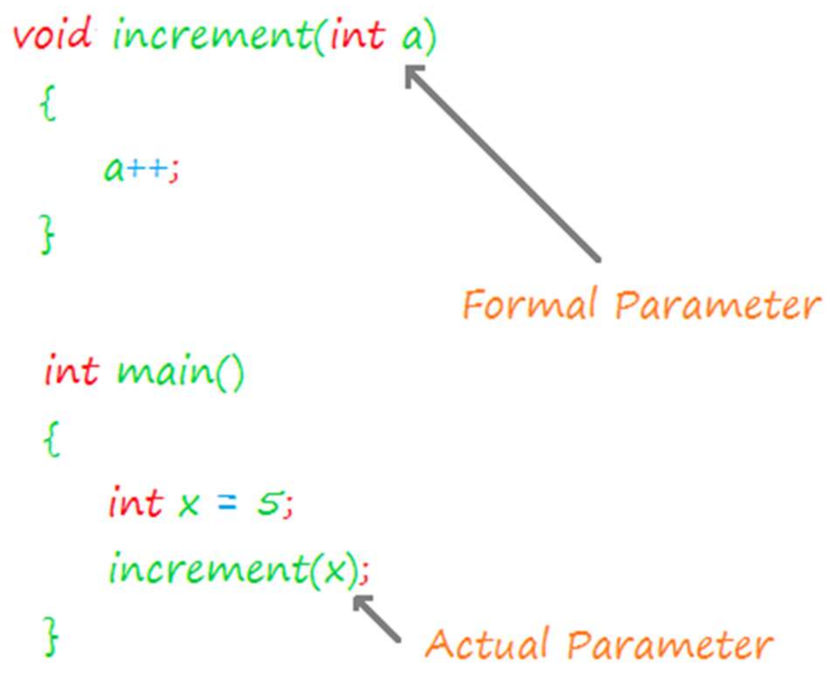
- Encapsulation is all about binding the data variables and functions together in class.

- Inheritance is a way to reuse once written code again and again.

- Polymorphism is to create functions with same name but different arguments, which will perform different actions.

C++ Functions : Recall

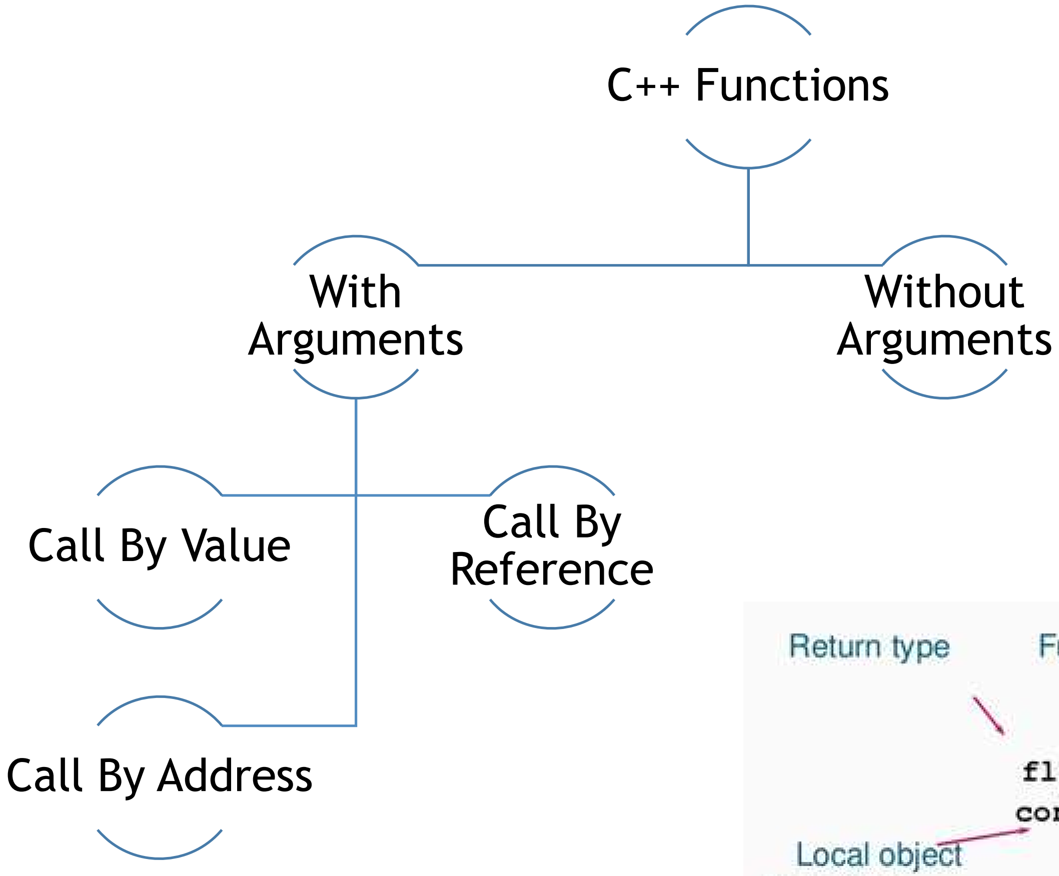
Difference between Formal and Actual Parameters

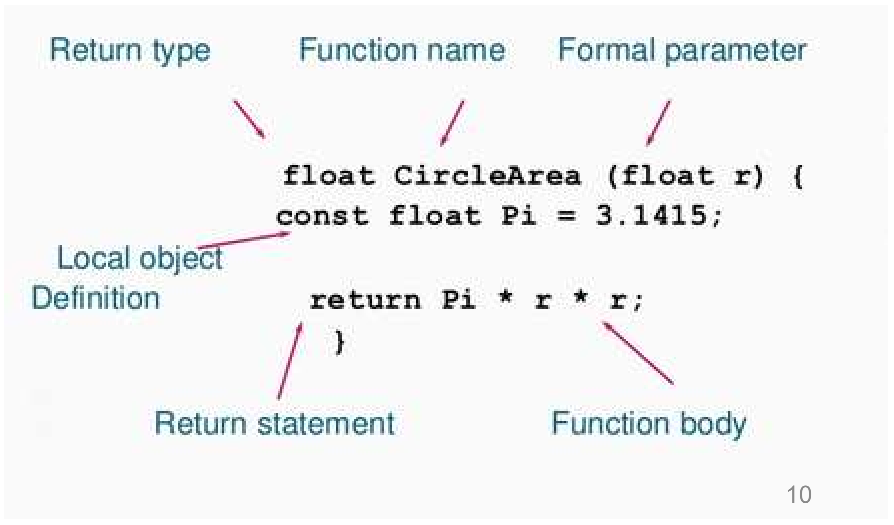


```
void increment(int a)
 {
     a++;
 }

 int main()
 {
     int x = 5;
     increment(x);
 }
```

C++ Functions



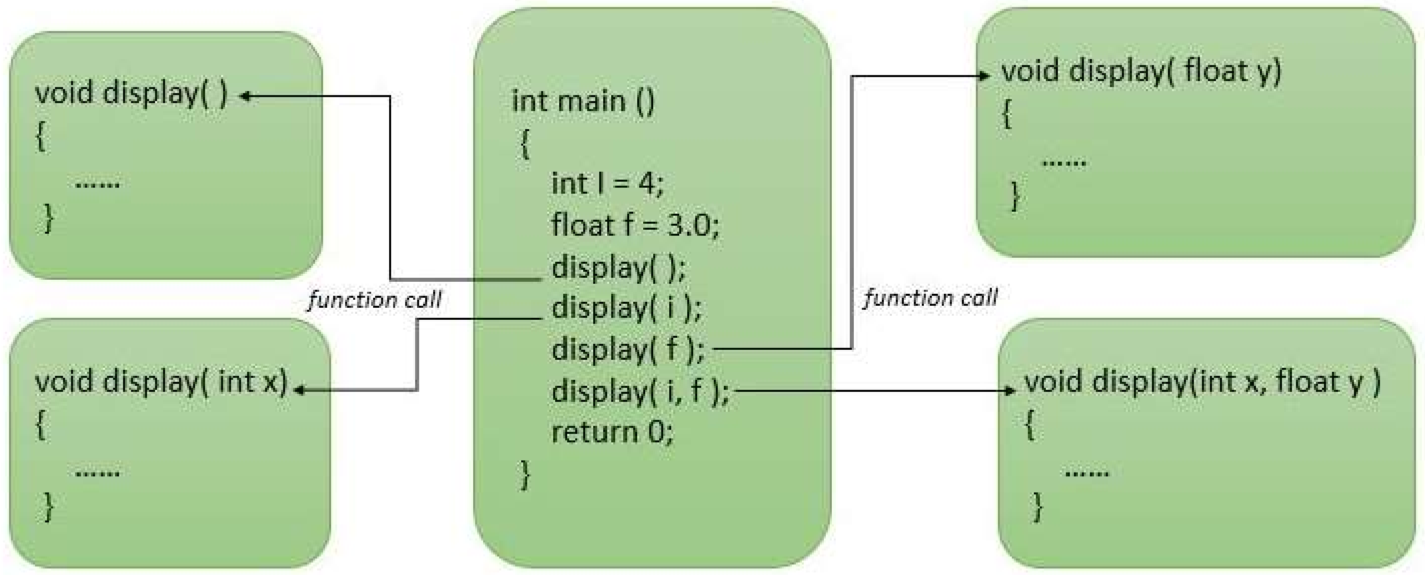


```
float CircleArea (float r) {
const float Pi = 3.1415;

 return Pi * r * r;
 }
```

Overloading

- Either an operator or a function can be redefined in C++.
- Function overloading is a feature in C++ where two or more functions can have the same name but different parameters.
- This is an instance of Polymorphism



```
void display( )
{
    ......
}
```

```
void display( int x)
{
    ......
}
```

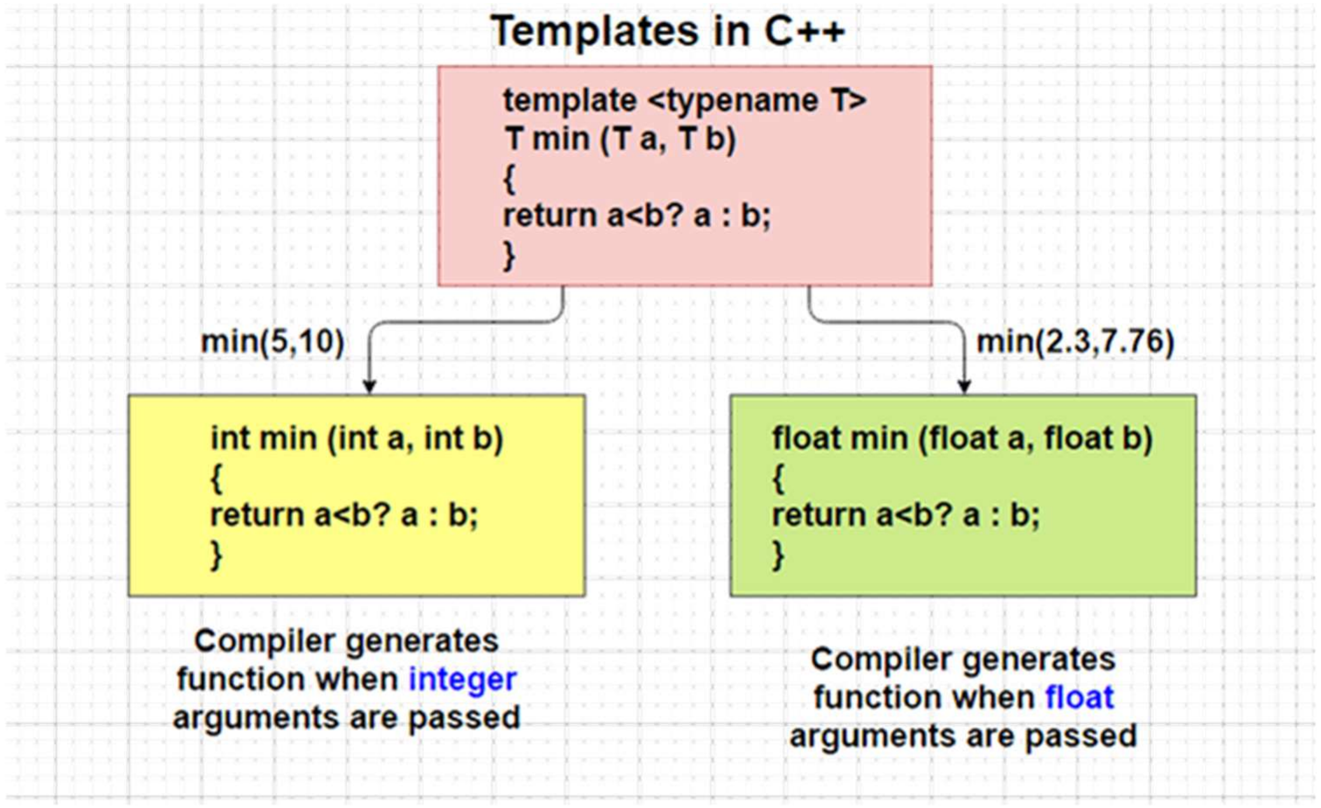
```
int main ()
{
    int I = 4;
    float f = 3.0;
    display( );
    display( i );
    display( f );
    display( i, f );
    return 0;
}
```

```
void display( float y)
{
    ......
}
```

```
void display(int x, float y )
{
    ......
}
```

Templates in C++

A template is a blueprint or formula for creating a generic class or a function.


Templates in C++
min(5,10)
min(2.3,7.76)
Compiler generates function when integer arguments are passed
Compiler generates function when float arguments are passed

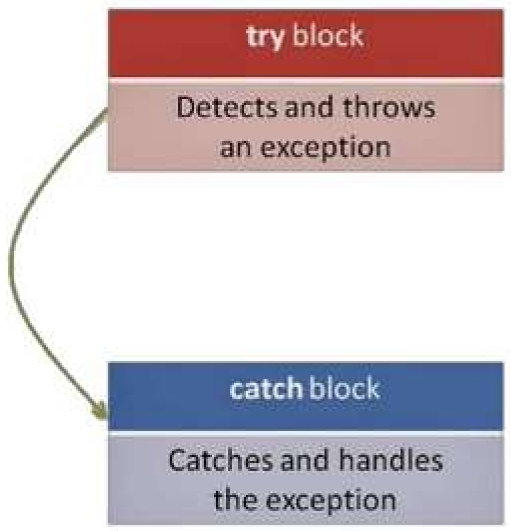
```
template <typename T>
T min (T a, T b)
{
return a<b? a : b;
}
```

```
int min (int a, int b)
{
return a<b? a : b;
}
```

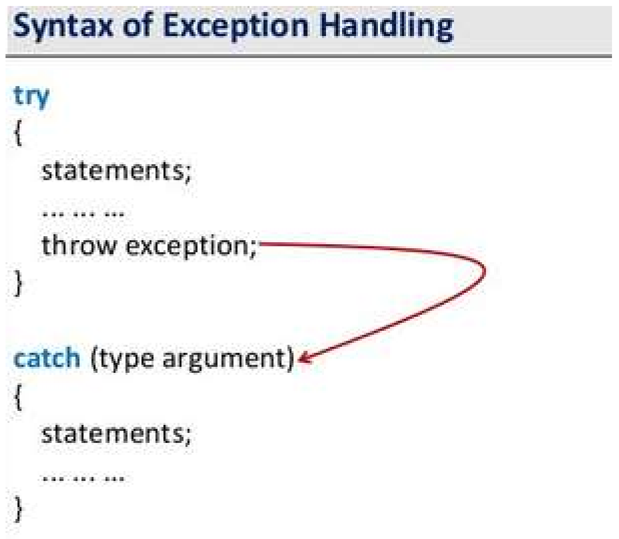
```
float min (float a, float b)
{
return a<b? a : b;
}
```

Exception Handling

- An exception is a problem that arises during the execution of a program.
- A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.
- Exceptions provide a way to transfer control from one part of a program to another.



try block
Detects and throws an exception
catch block
Catches and handles the exception



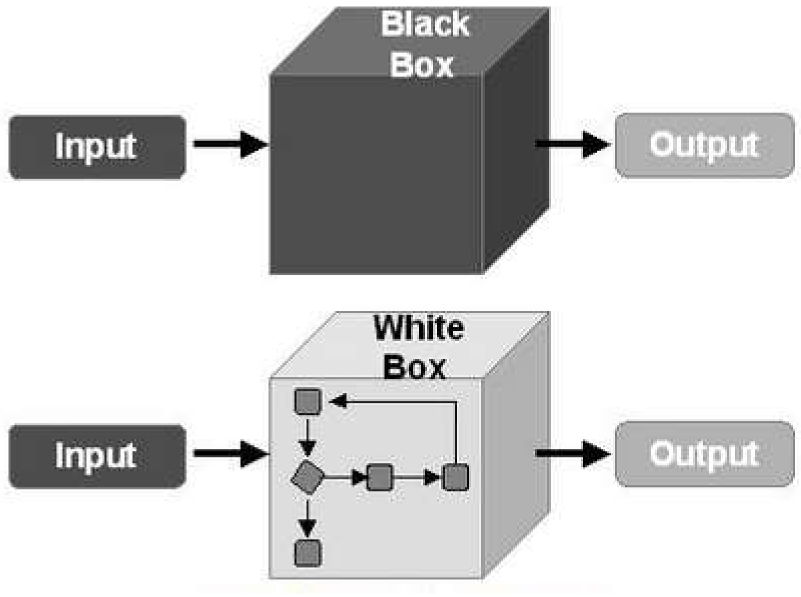
Syntax of Exception Handling

```
try
{
   statements;
   ... ... ...
   throw exception;
}

catch (type argument)
{
   statements;
   ... ... ...
}
```

Testing and Debugging

- Execute a program on target computer using input data (called test data) and compare the program's behavior with the expected behavior.



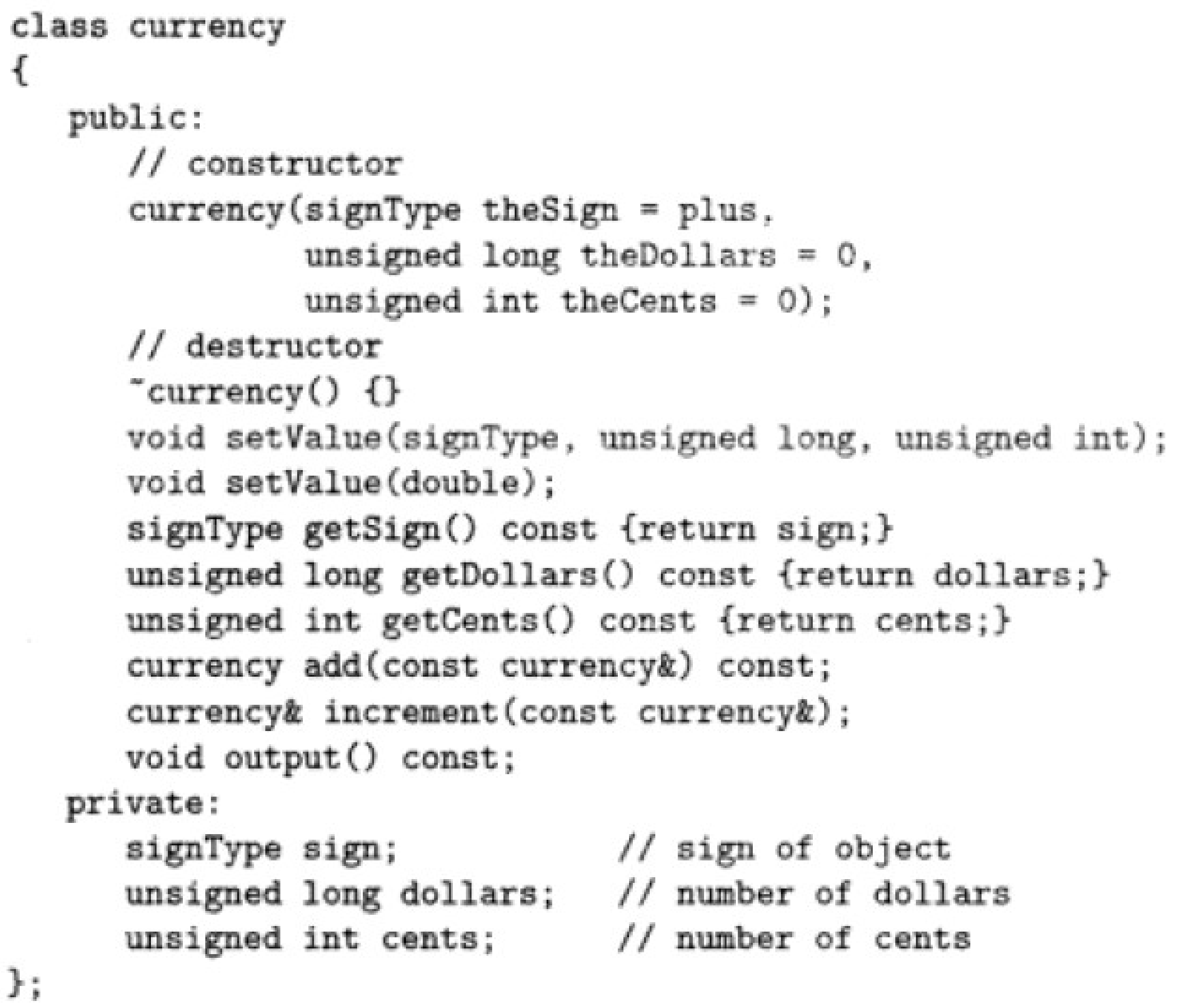
Revisiting C++ concepts: Sorting

Methods to sort the elements of an array {a[0] …. A[n-1]} in ascending / descending order

- Insertion Sort
- Bubble Sort
- Selection Sort
- Count Sort
- Shaker Sort
- Shell Sort
- Heap Sort
- Merge Sort
- Quick Sort

Find the animation of some of these sorting techniques here : https://visualgo.net/bn/sorting

Example to declare class : ‘currency’



```
class currency
{
   public:
      // constructor
      currency(signType theSign = plus,
               unsigned long theDollars = 0,
               unsigned int theCents = 0);
      // destructor
      ~currency() {}
      void setValue(signType, unsigned long, unsigned int);
      void setValue(double);
      signType getSign() const {return sign;}
      unsigned long getDollars() const {return dollars;}
      unsigned int getCents() const {return cents;}
      currency add(const currency&) const;
      currency& increment(const currency&);
      void output() const;
   private:
      signType sign;           // sign of object
      unsigned long dollars;   // number of dollars
      unsigned int cents;      // number of cents
};
```

Insert An Element

- Given a sorted list/sequence, insert a new element
- Given 3, 6, 9, 14
- Insert 5
- Result 3, 5, 6, 9, 14

Insert an Element

- 3, 6, 9, 14   insert 5
- Compare new element (5) and last one (14)
- Shift 14 right to get 3, 6, 9, , 14
- Shift 9 right to get 3, 6, , 9, 14
- Shift 6 right to get 3, , 6, 9, 14
- Insert 5 to get 3, 5, 6, 9, 14

Insert An Element

```
// insert t into a[0:i-1]
int j;
for (j = i - 1; j >= 0 && t < a[j]; j--)
   a[j + 1] = a[j];
a[j + 1] = t;
```

Insertion Sort

- Start with a sequence of size 1
- Repeatedly insert remaining elements

Insertion Sort

- Sort 7, 3, 5, 6, 1
- Start with 7 and insert 3 => 3, 7
- Insert 5 => 3, 5, 7
- Insert 6 => 3, 5, 6, 7
- Insert 1 => 1, 3, 5, 6, 7

Insertion Sort

```
for (int i = 1; i < a.length; i++)
{// insert a[i] into a[0:i-1]
   // code to insert comes here
}
```

Insertion Sort

```
for (int i = 1; i < a.length; i++)
{// insert a[i] into a[0:i-1]
   int t = a[i];
   int j;
   for (j = i - 1; j >= 0 && t < a[j]; j--)
       a[j + 1] = a[j];
   a[j + 1] = t;
}
```

Program Performance

- Program performance is the amount of computer memory and time needed to run a program.

- How is it determined?

1. Analytically
    - performance analysis
2. Experimentally
    - performance measurement

Criteria for Measurement

- Space
  - amount of memory program occupies
  - usually measured in bytes, KB or MB
- Time
  - execution time
  - usually measured by the number of executions

Space Complexity

- Space complexity is defined as the amount of memory a program needs to run to completion.

- Why is this of concern?
  - We could be running on a multi-user system where programs are allocated a specific amount of space.
  - We may not have sufficient memory on our computer.
  - There may be multiple solutions, each having different space requirements.
  - The space complexity may define an upper bound on the data that the program can handle.

Components of Program Space

- Program space = Instruction space + data space + stack space

- The instruction space is dependent on several factors.
  - the compiler that generates the machine code
  - the compiler options that were set at compilation time
  - the target computer

Components of Program Space

- Data space
  - very much dependent on the computer architecture and compiler
  - The magnitude of the data that a program works with is another factor

char	1	float	4
short	2	double	8
int	2	long double	10
long	4	pointer	2

Unit: bytes

Time Complexity

- Time complexity is the amount of computer time a program needs to run.

- Why do we care about time complexity?
  - Some computers require upper limits for program execution times.
  - Some programs require a real-time response.
  - If there are many solutions to a problem, typically we'd like to choose the quickest.

Time Complexity

- How do we measure?
  1. Count a particular operation (operation counts)
  2. Count the number of steps(step counts)
  3. Asymptotic complexity

Operation count ~ Comparison

- Pick an instance characteristic … n,

  Ex: n = a.length (number of elements) for insertion sort
- Determine count as a function of this instance characteristic.

```
for (int i = 1; i < a.length; i++)
{// insert a[i] into a[0:i-1]
   int t = a[i];
   int j;
   for (j = i - 1; j >= 0 && t < a[j]; j--)
      a[j + 1] = a[j];
   a[j + 1] = t;
}
```

Comparison Count

```
for (j = i - 1; j >= 0 && t < a[j]; j--)
      a[j + 1] = a[j];
```

How many comparisons are made?

- number of compares depends on a[] and t as well as on i

Comparison Count

- Worst-case count = maximum count
- Best-case count = minimum count
- Average count

Worst-Case Comparison Count

```
for (j = i - 1; j >= 0 && t < a[j]; j--)
        a[j + 1] = a[j];
```

- a = [1, 2, 3, 4] and t = 0 => 4 compares
- a = [1,2,3,…,i] and t = 0 => i compares

Worst-Case Comparison Count

```
for (int i = 1; i < n; i++)
   for (j = i - 1; j >= 0 && t < a[j]; j--)
       a[j + 1] = a[j];
```

- total compares = $1 + 2 + 3 + \ldots + (n-1)$

$$= (n-1)n/2$$

Step Count

- The operation count method omits accounting for the time spent on all but chosen operations.
- Step count method : account for the time spent in all parts of the program/method.
- A step is any computation unit that is independent of the selected characteristics.
- 10 adds, 100 subtracts, 1000 multiplies can all be counted as a single step
- n adds cannot be counted as 1 step (n is an instance characteristic)

- The amount of computing represented by one step may be different from that represented by another.

- For example, the entire statement

```
return a+b+b*c+(a+b-c)/(a+b)+4;
```

can be regarded as a single step if its execution time is independent of the problem size.

- We may also count a statement such as

`x = y;` as a single step.

Step Count

To determine the step count of an algorithm, we first determine the number of steps per execution (s/e) of each statement and the total number of times (i.e., frequency) each statement is executed.

```
                                              steps/execution (s/e)
for (int i = 1; i < n; i++)                         1
{                                                   0
  // insert a[i] into a[0:i-1]                      0
  int t = a[i];                                     1
  int j;                                            0
  for (j = i - 1; j >= 0 && t < a[j]; j--)          1
        a[j + 1] = a[j];                            1
  a[j + 1] = t;                                     1
}                                                   0
```

Step Count

The s/e of a statement is the amount by which Step Count changes as a result of execution of that statement.

s/e isn't always 0 or 1

x = sum(a, n);

where n is the instance characteristic and sum adds a[0:n-1] has a s/e count of n

Step Count

	s/e	frequency
for (int i = 1; i < n; i++)	1	n-1
{	0	0
// insert a[i] into a[0:i-1]	0	0
int t = a[i];	1	n-1
int j;	0	0
for (j = i - 1; j >= 0 && t < a[j]; j--)	1	(n-1)n/2
a[j + 1] = a[j];	1	(n-1)n/2
a[j + 1] = t;	1	n-1
}	0	n-1

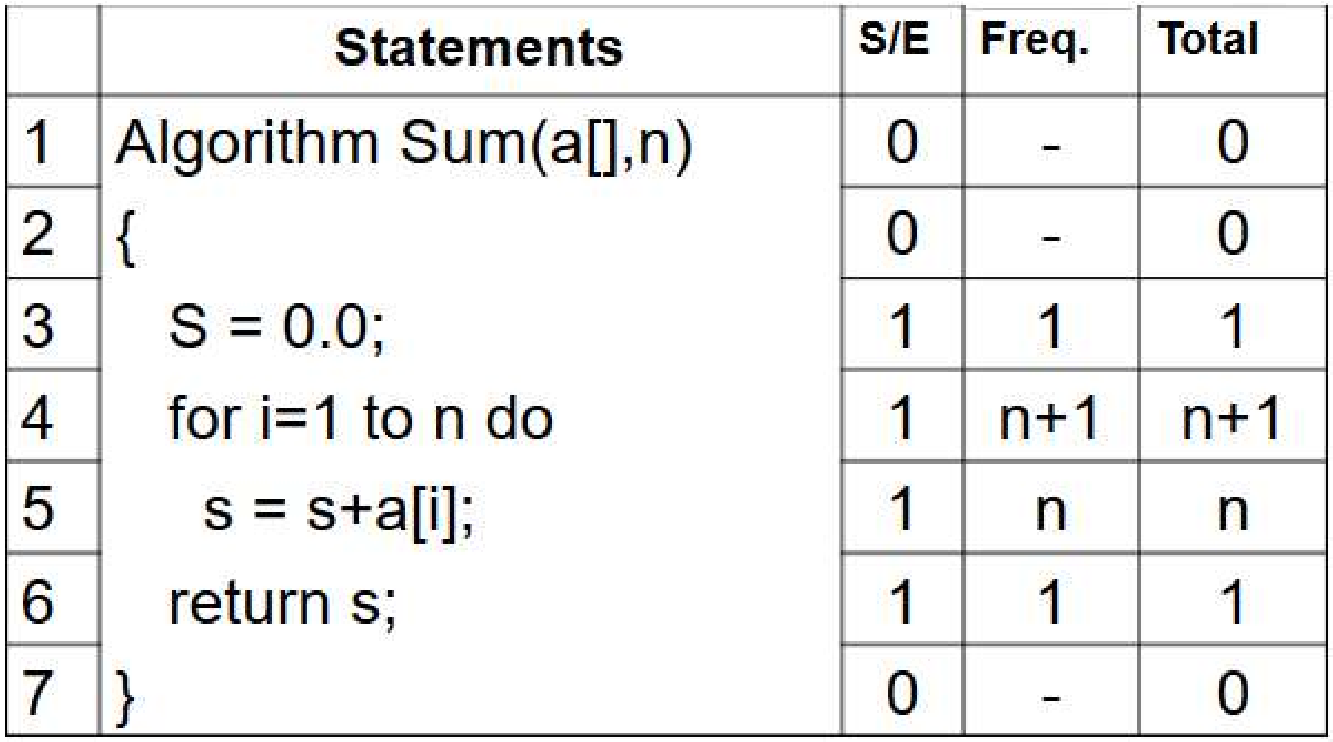
Step Count

Total step counts

$= (n-1) + 0 + 0 + (n-1) + 0 + (n-1)n/2 + (n-1)n/2 + (n-1) + (n-1)$

$= n^2 + 3n - 4$

Another example : Step Count

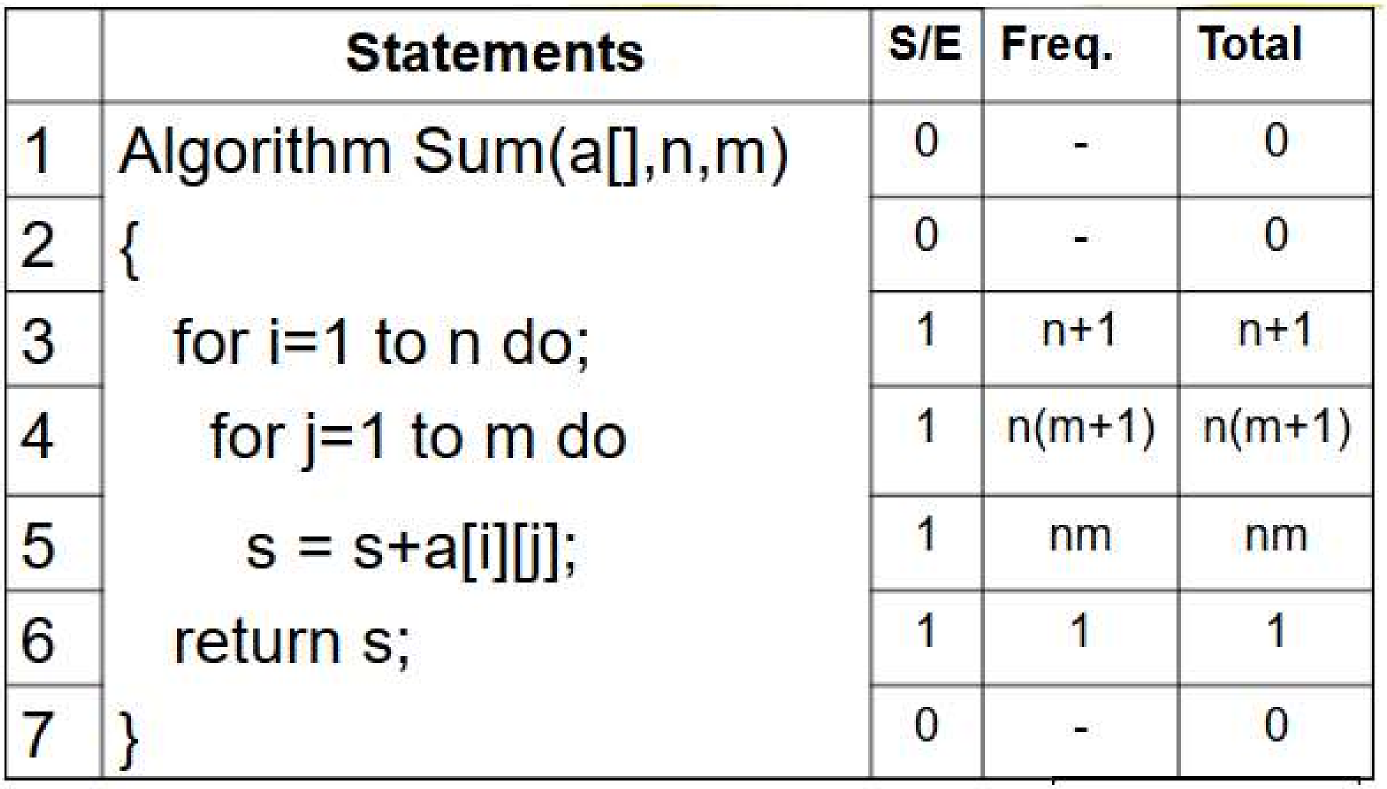


	Statements	S/E	Freq.	Total
1	Algorithm Sum(a[],n)	0	-	0
2	{	0	-	0
3	S = 0.0;	1	1	1
4	for i=1 to n do	1	n+1	n+1
5	s = s+a[i];	1	n	n
6	return s;	1	1	1
7	}	0	-	0

Total step counts

$= 2n + 3$

Another Example: Step Count



	Statements	S/E	Freq.	Total
1	Algorithm Sum(a[],n,m)	0	-	0
2	{	0	-	0
3	for i=1 to n do;	1	n+1	n+1
4	for j=1 to m do	1	n(m+1)	n(m+1)
5	s = s+a[i][j];	1	nm	nm
6	return s;	1	1	1
7	}	0	-	0

Total step counts

= 2(nm+n+1)

Faster Computer Vs Better Algorithm

- Algorithmic improvement more useful than hardware improvement.