**C++**

OOP – class, objects, encapsulation, polymorphism, inheritance, abstraction.

**Vectors**

* Store elements of similar data
* Size can grow dynamically (change size during exec of prog)
* Part of Standard Template library\*

STL – set of cpp template classes –provides common programming data structures and functions such as lists, stacks, array.

* #include<vector>
* **Declaration**: vector<int> n;
* **Initialization**

M1 – vector<int> n {1, 2, 3};

M2 – vector<int> n(3, 0); => vector<int> n{0,0,0}; // 3 – size of vector

* **Operations**

1. push\_back() – add single element to the end of a vector

2. access elements - .at() or [] but at() preferred bcoz it throws an exception whenever vector is out of bound unlike [] which gives a garbage value.

3. pop\_back() – delete elements from a vector.

* **Functions**

size(), clear(), front(), back(), empty(), capacity()

**size vs capacity**

The size of a vector represents the number of components in the vector.

The capacity of a vector represents the maximum number of elements the vector can hold.

**Vector iterators**

Vector iterators are used to point to the memory address of a vector element. In some ways, they act like [pointers](https://www.programiz.com/cpp-programming/pointers) in C++.

**Syntax**: vector<datatype>::iterator name;

**Initialize**

1. begin() - returns an iterator that points to the first element of the vector.
2. end() -  points to the **theoretical element** that comes **after** the final element of the vector.

Iterate through vector using iterators.

**Program**:

#include<iostream>

#include<vector>

using namespace std;

int main()

{

// vector initialization

vector<int> num{1,2,3};

// for loop to display vector items

for(int i=0;i<num.size();i++)

cout<<num[i];

cout<<endl;

// vector operations - add, delete, access

num.push\_back(7);

cout<<num[num.size()-1]<<endl;

num.pop\_back();

cout<<num[num.size()-1]<<endl;

cout<<"Element at 2nd position: "<<num.at(1)<<endl;

// for each loop to display vector items

for(int i: num)

cout<<i;

cout<<endl;

// vector iterators

vector<int> no{1, 2, 7, 5};

vector<int>::iterator iter;

for(iter = no.begin(); iter!= no.end(); iter++)

cout<<\*iter;

// initialize iterators using begin() or end()

iter = no.begin();

cout<<"1st element: "<<\*iter<<endl;

iter = no.end();

cout<<"last element: "<<\*iter<<endl;

return 0;

}

**Templates**

* write generic programs – types to-be-specified-later
* implement templates: function, class – to create a single class to work with different data types.
* Declaration template<class T> // T – placeholder for the data type used

Templates with multiple parameters

* We can use multiple template parameters
* We can use default arguments
* Syntax: template <class T, class U, class V = int> // v – default parameter int

**Program:**

#include <iostream>

using namespace std;

template <class T>

class Calculator

{

private:

T num1, num2;

public:

template <typename F>

void check(F no)

{

cout<<no<<endl;

}

void displayResult(T num1, T num2)

{

cout << "Numbers: " << num1 << " and " << num2 << "." << endl;

cout << num1 << " + " << num2 << " = " << add(num1, num2) << endl;

cout << num1 << " - " << num2 << " = " << subtract(num1, num2) << endl;

cout << num1 << " \* " << num2 << " = " << multiply(num1, num2) << endl;

cout << num1 << " / " << num2 << " = " << divide(num1, num2) << endl;

}

T add(T num1, T num2) { return num1 + num2; }

T subtract(T num1, T num2) { return num1 - num2; }

T multiply(T num1, T num2) { return num1 \* num2; }

T divide(T num1, T num2) { return num1 / num2; }

};

int main()

{

Calculator<int> obj;

obj.displayResult(2,1);

Calculator<int> obj1;

obj1.check<float>(3.8);

// All functions using T will be assigned with int and

// those using F with float

return 0;

}

**Constructors**

* Called automatically when an object is created
* Same name as that of class
* Has no return type
* Uses: to initialize objects, also to run a default code when an object is created.
* **Types** 
  + **Default constructor** – no parameters
  + **Parameterized constructor** – with param

Code:

class1(int first, int second)

{

a = first;

b = second;

}

main()

{

class1 obj(1,2); // pass values to the parameterized constructor

}

* + **Copy constructor** - copy data of one object to another

Code:

class1(class1 &obj)

{

a = obj.a;

b = obj.b;

}