**Typrcasting**

#include <iostream>

using namespace std;

int main() {

double a = 21.099;

float b = 10.34;

int c;

c = (int) a;

cout<< "line 1 - value of (int) a is :"<<c<<endl;

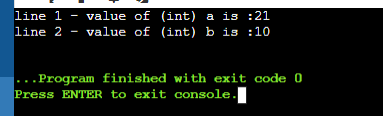
c = (int) b;

cout<<"line 2 - value of (int) b is :"<<c<<endl;

return 0;

}

Output:



#include <iostream>

using namespace std;

int main() {

int x = 10;

char y = 'a';

x = x+y;

float z = x + 1.0;

cout<<" x "<<x<<endl;

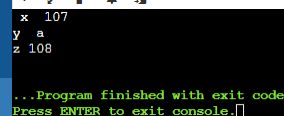
cout<<"y "<<y<<endl;

cout<<"z "<<z<<endl;

return 0;

}

Output:



#include <iostream>

using namespace std;

int main() {

double x =1.2;

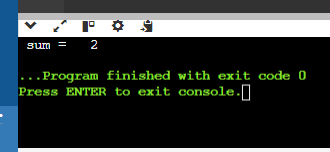
int sum = (int)x + 1;

cout<<" sum = "<<sum;

return 0;

}

Output:



#include <iostream>

using namespace std;

int main(){

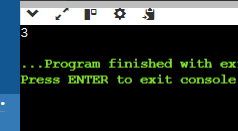
float f = 3.5;

int b = static\_cast<int>(f);

cout<< b;

}

Output:



#include <iostream>

#include <typeinfo> // for dynamic\_cast

class Base {

public:

virtual void whoami() {

std::cout << "I am a Base class object" << std::endl;

}

virtual ~Base() {} // Adding virtual destructor for polymorphic behavior

};

class Derived : public Base {

public:

void whoami() override {

std::cout << "I am a Derived class object" << std::endl;

}

};

int main() {

// static\_cast example (truncating double to int)

double num = 3.14159;

int integer\_part = static\_cast<int>(num); // Truncates the decimal

std::cout << "Original number: " << num << std::endl;

std::cout << "Integer part: " << integer\_part << std::endl;

// Incorrect upcasting (assuming Derived object but not guaranteed)

// This could lead to undefined behavior if base\_ptr doesn't point to a Derived

Base\* base\_ptr = new Derived(); // Pointer to a Base class (actually pointing to a Derived)

Derived\* derived\_ptr = static\_cast<Derived\*>(base\_ptr); // Attempting to cast to Derived\*

// Safer approach: check the actual type before downcasting

if (dynamic\_cast<Derived\*>(base\_ptr) != nullptr) {

derived\_ptr = static\_cast<Derived\*>(base\_ptr); // Downcast only if safe

derived\_ptr->whoami(); // Call Derived class's whoami

} else {

std::cout << "Warning: Base object might not be of type Derived" << std::endl;

}

delete base\_ptr; // Release memory

// Using reinterpret\_cast example

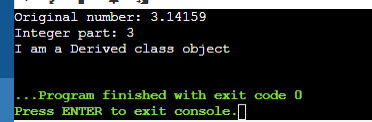
int value = 10;

float\* float\_ptr = reinterpret\_cast<float\*>(&value);

return 0;

}

Output:



#include <iostream>

using namespace std;

int main() {

vector<int> vec;

int i;

cout<<"vector size "<< vec.size()<< endl;

for(i=0;i<5;i++)

{

vec.push\_back(i);

}

cout<<"extended vector size "<<vec.size() << endl;

for(i = 0; i<5; i++)

{

cout<<"value of vec["<< i << "] "<< vec[i]<<endl;

}

vector<int>::iterator v = vec.begin();

while(v != vec.end())

{

cout<<"value of v "<< \*v<< endl;

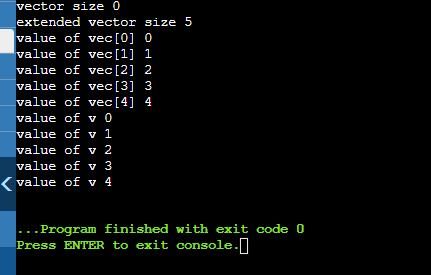
v++;

}

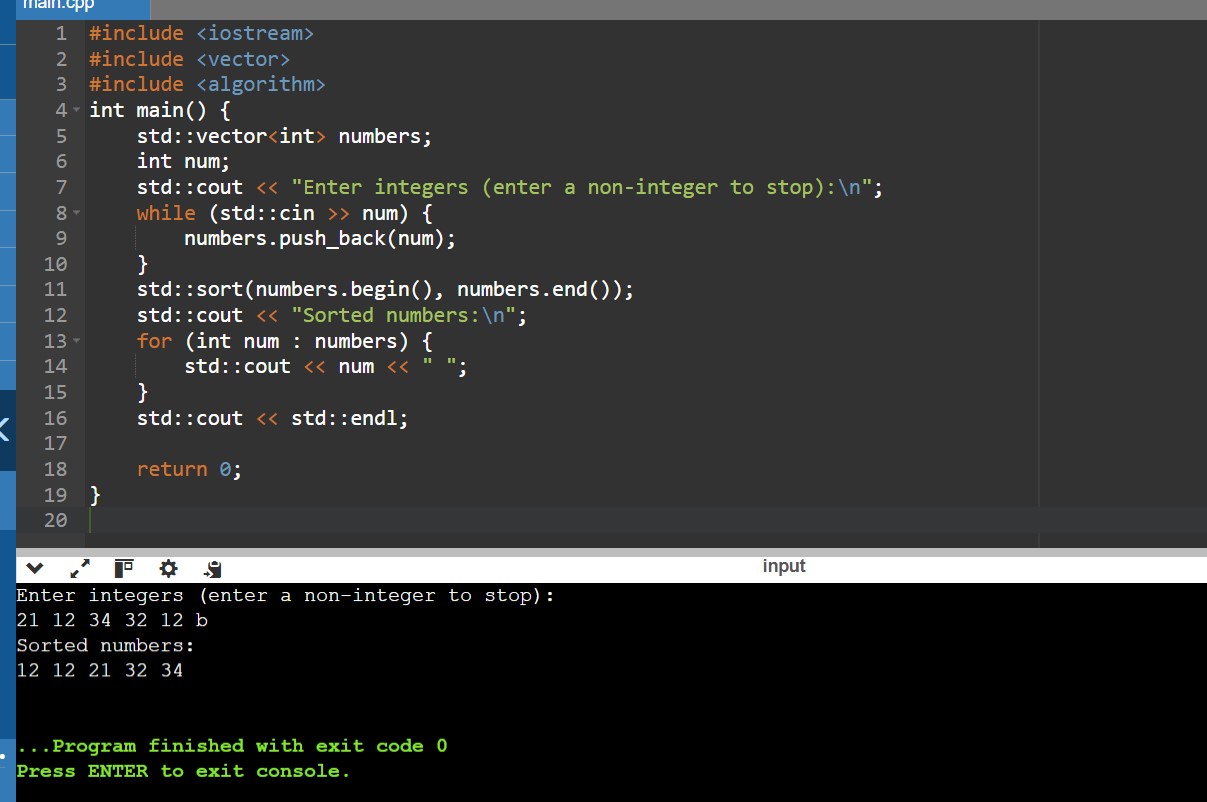
return 0;

}

Output:



Using Vector:



Without Using Vector:

