**Programs:**

1. Bubble sort using Function templates.

#include <conio.h>

#include <iostream>

using namespace std;

//Declaration of template class bubble

template <class bubble>

void bubbleSort(bubble a[], int n)

{

int i, j;

for (i = 0; i < n - 1; i++)

{

for (j = i + 1; j < n; j++)

{

if (a[i] > a[j])

{

bubble b;

b = a[i];

a[i] = a[j];

a[j] = b;

}

}

}

}

int main()

{

int arr[20], k, i;

char ch[20];

cout << "\nEnter the number of elements in integer array:";

cin >> k;

cout << "\nEnter elements:";

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

cin >> arr[i];

bubbleSort(arr, k);

cout << "\nSorted integer array: ";

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

cout << arr[i] << "\t";

cout << "\nEnter the number of characters in the array:";

cin >> k;

cout << "\nEnter elements:";

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

cin >> ch[i];

bubbleSort(ch, k);

cout << "\nSorted character array: ";

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

cout << ch[i] << "\t";

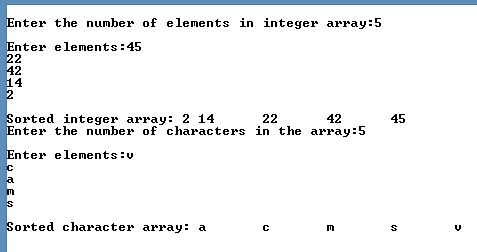
cout << endl;

getch();

return 0;

}

**Output:**



1. Function overloading of display(). The 3 forms will be
   1. Displaying 2 numbers of different types
   2. Displaying 1 template type variable and 1 built-in type

#include <iostream>

using namespace std;

template <typename T>

void print(T num1){

cout << "Number : " << num1 << endl;

}

template <typename T>

void print(T num1, int inNum){

cout << "Number 1: " << num1 << endl;

cout << "Number 2: " << inNum << endl;

}

int main() {

int inNum;

float flNum;

cout << "Enter a number of interger type : ";

cin >> inNum;

print(inNum);

cout << "Enter a number of float type : ";

cin >> flNum;

print(flNum);

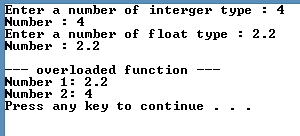
cout << endl << "--- overloaded function ---" << endl;

print(flNum, inNum);

return 0;

}

**Output:**



1. Program to add, subtract, multiply and divide two numbers using class template.

#include <iostream>

using namespace std;

template <class T>

class Calculator {

private:

T num1, num2;

public:

Calculator(T n1, T n2) {

num1 = n1;

num2 = n2;

}

void displayResult() {

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

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

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

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

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

}

T add() { return num1 + num2; }

T subtract() { return num1 - num2; }

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

T divide() { return num1 / num2; }

};

int main() {

Calculator<int> intNums(40, 10);

Calculator<float> floatNums(4.4, 2.2);

cout << "----- Integer -----" << endl;

intNums.displayResult();

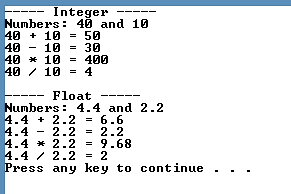
cout << endl << "----- Float -----" << endl;

floatNums.displayResult();

return 0;

}

**Output:**



1. Define class Stack<> and implement generic methods to push and pop the elements from the stack

#include <iostream>

#include <string>

using namespace std;

#define SIZE 5

template <class T> class Stack {

public:

Stack();

void push(T k);

T pop();

T topElement();

bool isFull();

bool isEmpty();

private:

int top;

T st[SIZE];

};

template <class T> Stack<T>::Stack() { top = -1; }

template <class T> void Stack<T>::push(T k)

{

if (isFull()) {

cout << "Stack is full\n";

}

cout << "Inserted element " << k << endl;

top += 1;

st[top] = k;

}

template <class T> bool Stack<T>::isEmpty()

{

if (top == -1)

return 1;

else

return 0;

}

template <class T> bool Stack<T>::isFull()

{

if (top == (SIZE - 1))

return 1;

else

return 0;

}

template <class T> T Stack<T>::pop()

{

T popped\_element = st[top];

top--;

return popped\_element;

}

template <class T> T Stack<T>::topElement()

{

T top\_element = st[top];

return top\_element;

}

int main()

{

Stack<int> integer\_stack;

Stack<string> string\_stack;

cout << "----- Interger Stack -----" << endl;

integer\_stack.push(2);

integer\_stack.push(54);

integer\_stack.push(255);

cout << integer\_stack.pop() << " is popped from stack" << endl;

cout << "Top element is " << integer\_stack.topElement() << endl;

cout << "\n----- String Stack -----" << endl;

string\_stack.push("Hello");

string\_stack.push("world");

cout << string\_stack.pop() << " is popped from stack " << endl;

cout << "Top element is " << string\_stack.topElement() << endl;

return 0;

}

**Output:**

