

# Programming in C++: Templates

## Function Templates

// tempabs.cpp -- template used for absolute value function  
#include <iostream.h>

```
template <class T> // function template
T abs(T n)
{
return (n < 0) ? -n : n;
}
```

```
void main()
{
int n1 = 5;
int n2 = -6;
long l1 = 70000L;
long l2 = -80000L;
double d1 = 9.95;
double d2 = -10.15;

// compiler sees function call for a particular data type, creates
// another machine-language instance of function.
cout << "\nabs(" << n1 << ")=" << abs(n1); // abs(int)
cout << "\nabs(" << n2 << ")=" << abs(n2); // abs(int)
cout << "\nabs(" << l1 << ")=" << abs(l1); // abs(long)
cout << "\nabs(" << l2 << ")=" << abs(l2); // abs(long)
cout << "\nabs(" << d1 << ")=" << abs(d1); // abs(double)
cout << "\nabs(" << d2 << ")=" << abs(d2); // abs(double)
cout << endl;
}
```

// tempfind.cpp  
// template used for function that finds number in array  
#include <iostream.h>

```
// function returns index number of item, or -1 if not found
template <class atype>
int find(atype* array, atype value, int nSize)
{
for (int j=0; j<nSize; j++)
if (array[j] == value)
return j;
return -1;
}
```

```
void main()
{
char acArr[ ] = {1, 3, 5, 9, 11, 13}; // array
char c = 5; // value to find
int anArr[ ] = {1, 3, 5, 9, 11, 13};
int n = 6;
long alArr[ ] = {1L, 3L, 5L, 9L, 11L, 13L};
long l = 11L;
double adArr[ ] = {1.0, 3.0, 5.0, 9.0, 11.0, 13.0};
double d = 4.0;

cout << "\n 5 in acArr: index=" << find(acArr, c, 6);
cout << "\n 6 in anArr: index=" << find(anArr, n, 6);
cout << "\n 11 in alArr: index=" << find(alArr, l, 6);
cout << "\n 4 in adArr: index=" << find(adArr, db, 6);
cout << endl;
}
```

## Class Templates: Array Container

### ArrayTemplate.hpp

```

#if ! defined _ARRAYTEMPLATE_HPP_ // guard against
#define _ARRAYTEMPLATE_HPP_ // multiple includes
template <class TYPE>
class Array {
private:
    TYPE** ppData; // TYPE is variable data type
    int nSize;
    int nNum;

public:
    Array(int n) : nSize(n), ppData(new TYPE*[n]), nNum(0) {}
    ~Array() { DeleteAll(); delete [ ] ppData; }

    bool AddItem(TYPE* pNewData)
    {
        if (nNum < nSize) {
            ppData[nNum++] = pNewData;return true;
        }
        else
            return false;
    }

    void DeleteAll()
    {
        for (int i=0; i<nNum; ++i)
            delete ppData[i]; // delete each object one at a time
        nNum = 0;
    }

    void DisplayAll()
    {
        for (int i=0; i<nNum; ++i)
            cout << *ppData[i]; // challenge to implement
    }
};
#endif // _ARRAYTEMPLATE_HPP_

```

## Sample Code to Test Array Template

```

#include <iostream>
using namespace std;
#include "ArrayTemplate.hpp"

```

```

class Student {
private:
    char szName[20];
    int nGrade;

public:
    Student()
    { cout << "Name: "; cin >> szName;
      cout << "Grade: "; cin >> nGrade; }

    friend ostream& operator<<(ostream& ros, Student& rs);
};

```

```

ostream& operator<<(ostream& ros, Student& rs)
{
    cout << "Name: " << rs.szName << " Grade: " << rs.nGrade;
    return ros;
}

```

```

void main()
{
    int nSize = 50;
    Array<Student> asStudents(nSize);
    char cResponse;
    do {
        if ( ! asStudents.AddItem(new Student))
            break;
        cout << "Continue (y/n): "; cin >> cResponse;
    } while (cResponse == 'y');

    asStudents.DisplayAll();
    // all memory cleaned up by class destructor
}

```