Programming in Modern C++

Module M03: Arrays and Strings

NPTEL

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

All url's in this module have been accessed in September, 2021 and found to be functional

Module Recap

- Understanding differences between C and C++ for:
  - IO
  - Variable declaration
  - Standard Library
  - bool
- C++ gives us more flexibility in terms of basic declaration and input / output
- Many C constructs and functions are simplified in C++ which helps to increase the ease of programming

NPTEL

Module Objectives

- Understand array usage in C and C++
- Understand `vector` usage in C++
- Understand `string` functions in C and `string` type in C++

NPTEL

Module Outline

1. Arrays & vectors
   - Array Implementations for fixed size array
   - Array Implementations for arbitrary sized array
   - vectors in C++
2. C-Style Strings and string type in C++
   - Concatenation of strings
   - More string operations
   - string.h
   - string class
3. Module Summary

NPTEL

NPTEL

Arrays and vectors

Program 03.01: Fixed Size Array

C Program

```
// Array_Fixed_Size.c
#include <stdio.h>

int main() {
    short age[4];

    age[0] = 23;
    age[1] = 34;
    age[2] = 65;
    age[3] = 74;

    printf("%d ", age[0]);
    printf("%d ", age[1]);
    printf("%d ", age[2]);
    printf("%d ", age[3]);

    return 0;
}
```

```
23 34 65 74
```

C++ Program

```
// Array_Fixed_Size_c++.cpp
#include <iostream>

int main() {
    short age[4];

    age[0] = 23;
    age[1] = 34;
    age[2] = 65;
    age[3] = 74;

    std::cout << age[0] << " ";
    std::cout << age[1] << " ";
    std::cout << age[2] << " ";
    std::cout << age[3] << " ";

    return 0;
}
```

```
23 34 65 74
```

NPTEL

- No difference between arrays in C and C++

Program 03.02: Fixed size large array in C

Hard-coded

```
// Array_Large_Size.c
#include <stdio.h>
#include <stdlib.h>


int main() { int arr[100], sum = 0, i;
    printf("Enter no. of elements: ");
    int count;
    scanf("%d", &count);

    for(i = 0; i < count; i++) {
        arr[i] = i;
        sum + = arr[i];
    }
    printf("Array Sum: %d", sum);
}
```

```
Enter no. of elements: 10
Array Sum: 45
```

- Hard-coded size

Using manifest constant

```
// Array_Macro.c
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

int main() { int arr[MAX], sum = 0, i;
    printf("Enter no. of elements: ");
    int count;
    scanf("%d", &count);

    for(i = 0; i < count; i++) {
        arr[i] = i;
        sum + = arr[i];
    }
    printf("Array Sum: %d", sum);
}
```

```
Enter no. of elements: 10
Array Sum: 45
```

- Size by manifest constant

NPTEL

Arbitrary Size Array

This can be implemented in C (C++) in the following ways:

- **Case 1:** Declaring a large array with size greater than the size given by users in all (most) of the cases
  - Hard-code the maximum size in code
  - Declare a manifest constant for the maximum size
- **Case 2:** Using `malloc` (`new[]`) to dynamically allocate space at run-time for the array

NPTEL

Program 03.03: Fixed large array / vector

C (array & constant)

```c
// Array_Macro_c.c
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

int main() { int arr[MAX];
    printf("Enter no. of elements: ");
    int count, sum = 0, i;
    scanf("%d", &count);
    for(i = 0; i < count; i++) {
        arr[i] = i; sum + = arr[i];
    }
    printf("Array Sum: %d", sum);
}
```

```
Enter no. of elements: 10
Array Sum: 45
```

- MAX is the declared size of array
- No header needed
- arr declared as int []

C++ (vector & constant)

```cpp
// Array_Macro_c++.cpp
#include <iostream>
#include <vector>
using namespace std;
#define MAX 100

int main() { vector<int> arr(MAX); // Define-time size
    cout <<"Enter the no. of elements: ";
    int count, sum = 0;
    cin >>count;
    for(int i = 0; i < count; i++) {
        arr[i] = i; sum + = arr[i];
    }
    cout << "Array Sum: " << sum << endl;
}
```

```
Enter no. of elements: 10
Array Sum: 45
```

- MAX is the declared size of vector
- Header vector included
- arr declared as vector<int>

NPTEL

Program 03.04: Dynamically managed array size

C Program

```
// Array_Malloc.c
#include <stdio.h>
#include <stdlib.h>


int main() { printf("Enter no. of elements ");
    int count, sum = 0, i;
    scanf("%d", &count);

    int *arr = (int*) malloc
        (sizeof(int)*count);
    for(i = 0; i < count; i++) {
        arr[i] = i; sum + = arr[i];
    }
    printf("Array Sum:%d ", sum);
}
```

```
Enter no. of elements: 10
Array Sum: 45
```

- malloc allocates space using sizeof

C++ Program

```
// Array_Resize_c++.cpp
#include <iostream>
#include <vector>
using namespace std;

int main() { cout << "Enter the no. of elements: ";
    int count, sum=0;
    cin >> count;

    vector<int> arr;    // Default size
    arr.resize(count);  // Set resize
    for(int i = 0; i < arr.size(); i++) {
        arr[i] = i; sum + = arr[i];
    }
    cout << "Array Sum: " << sum << endl;
}
```

```
Enter no. of elements: 10
Array Sum: 45
```

- resize fixes vector size at run-time

NPTEL

NPTEL

C-Style Strings and string type in C++

Strings in C and C++

String manipulations in C and C++:

- C-String and `string.h` library
  - C-String is an array of `char` terminated by `NULL`
  - C-String is supported by functions in `string.h` in C standard library
- `string` type in C++ standard library
  - `string` is a type
  - With operators (like `+` for concatenation) it behaves like a built-in type
  - In addition, for functions from C Standard Library `string.h` can be used in C++ as `cstring` in `std` namespace

NPTEL

Program 03.05: Concatenation of Strings

C Program

```
// Add_strings.c
#include <stdio.h>
#include <string.h>


int main() { char str1[] = {'H','E','L','L','O',' ','\0'};
    char str2[] = "WORLD";
    char str[20];
    strcpy(str, str1);
    strcat(str, str2);

    printf("%s\n", str);
}
```

```
HELLO WORLD
```

- Need header `string.h`
- *C-String is an array of characters*
- String concatenation done with `strcat` function
- Need a copy into `str`
- `str` must be large to fit the result

C++ Program

```
// Add_strings_c++.cpp
#include <iostream>
#include <string>
using namespace std;

int main(void) { string str1 = "HELLO ";
    string str2 = "WORLD";


    string str = str1 + str2;

    cout << str;
}
```

```
HELLO WORLD
```

- Need header `string`
- `string` is a data-type in C++ standard library
- Strings are concatenated like addition of `int`

NPTEL

More Operations on Strings

Further,

- `operator=` can be used on strings in place of `strcpy` function in C
- `operator<=`, `operator<`, `operator>=`, `operator>` operators can be used on strings in place of `strcmp` function in C

NPTEL

Strings in C and C++: C Standard Library Functions

Function	Description	Used Frequently?
Copying: `memcpy`	Copy block of memory (function)	Yes
`memmove`	Move block of memory (function)	Yes
`strcpy`	Copy string (function)	Yes
`strncpy`	Copy characters from string (function)	
Concatenation: `strcat`	Concatenate strings (function)	Yes
`strncat`	Append characters from string (function)	
Comparison: `memcmp`	Compare two blocks of memory (function)	
`strcmp`	Compare two strings (function)	Yes
`strcoll`	Compare two strings using locale (function)	
`strncmp`	Compare characters of two strings (function)	
`strxfrm`	Transform string using locale (function)	
Searching: `memchr`	Locate character in block of memory (function)	Yes
`strchr`	Locate first occurrence of character in string (function)	Yes
`strcspn`	Get span until character in string (function)	
`strpbrk`	Locate characters in string (function)	
`strrchr`	Locate last occurrence of character in string (function)	
`strspn`	Get span of character set in string (function)	
`strstr`	Locate substring (function)	Yes
`strtok`	Split string into tokens (function)	Yes
Other: `memset`	Fill block of memory (function)	
`strerror`	Get pointer to error message string (function)	
`strlen`	Get string length (function)	Yes
Macros: `NULL`	Null pointer (macro)	Yes
Types: `size_t`	Unsigned integral type (type)	Yes

Proof

Strings in C and C++: C++ Standard Library `string` Class

- Strings are objects that represent sequences of characters
- The standard string class provides support for such objects with an interface similar to that of a standard container of bytes, but adding features specifically designed to operate with strings of single-byte characters
- The string class is an instantiation of the `basic_string` class template that uses `char` (that is, bytes) as its character type, with its default `char_traits` and `allocator` types

Function	Description (Member Function)	C Parallel
Member functions		
`(constructor)`	Construct string object (public)	Initialize `string` object with a C string
`(destructor)`	String destructor (public)	
`operator=`	String assignment (public)	`strcpy()`. `operator=` does shallow copy
Iterators		Iteration done explicitly by loop index
`begin`	Return iterator to beginning (public)	
`end`	Return iterator to end (public)	
`rbegin`	Return reverse iterator to reverse beginning (public)	
`rend`	Return reverse iterator to reverse end (public)	
`cbegin`	Return const_iterator to beginning (public)	
`cend`	Return const_iterator to end (public)	
`crbegin`	Return const_reverse_iterator to reverse beginning (public)	
`crend`	Return const_reverse_iterator to reverse end (public)	

NPTEL

Strings in C and C++: C++ Standard Library `string` Class

Function	Description (Member Function)	C Parallel
Capacity		
`size`	Return length of string (public)	`strlen()`
`length`	Return length of string (public)	`strlen()`
`max_size`	Return maximum size of string (public)	Fixed at allocation
`resize`	Resize string (public)	
`capacity`	Return size of allocated storage (public)	Need to be remembered in the code
`reserve`	Request a change in capacity (public)	
`clear`	Clear string (public)	`strcpy()` an empty string
`empty`	Test if string is empty (public)	`strlen() == 0`
`shrink_to_fit`	Shrink to fit (public)	
String operations		
`c_str`	Get C string equivalent (public)	C string from a `string` object
`data`	Get string data (public)	
`get_allocator`	Get allocator (public)	
`copy`	Copy sequence of characters from string (public)	`strncpy()`
`find`	Find content in string (public)	`strchr()`, `strstr()`
`rfind`	Find last occurrence of content in string (public)	
`find_first_of`	Find character in string (public)	`strchr()`
`find_last_of`	Find character in string from the end (public)	`strrchr()`
`find_first_not_of`	Find absence of character in string (public)	
`find_last_not_of`	Find non-matching character in string from the end (public)	
`substr`	Generate substring (public)	`strncpy()`
`compare`	Compare strings (public)	`strcmp()`

NPTEL

Strings in C and C++: C++ Standard Library `string` Class

Function	Description (Member Function)	C Parallel
Element access		
`operator[]`	Get character of string (public)	`operator[]`
`at`	Get character in string (public)	`operator[]`
`back`	Access last character (public)	Character at `strlen()-1`
`front`	Access first character (public)	Character at 0^{th} location
Modifiers		
`operator+=`	Append to string (public)	`strcat()`
`append`	Append to string (public)	`strcat()`
`push_back`	Append character to string (public)	Set character to `strlen()` and `NULL` to next location
`assign`	Assign content to string (public)	
`insert`	Insert into string (public)	
`erase`	Erase characters from string (public)	
`replace`	Replace portion of string (public)	
`swap`	Swap string values (public)	Character by character swapping between two arrays
`pop_back`	Delete last character (public)	Set location `strlen()-1` to `NULL`
Member constants		
`npos`	Maximum value for size_t (public static)	
Non-member function overloads		
`operator+`	Concatenate strings (global)	`strcat()`
relational operators	Relational operators for string (global)	`strcmp()` followed by tests for `-1`, `0`, `+1`
`swap`	Exchanges the values of two strings (global)	
`operator>>`	Extract string from stream (global)	format `%s`
`operator<<`	Insert string into stream (global)	format `%s`
`getline`	Get line from stream into string (global)	`getline()` in `<stdlib.h>`

NPTEL

Module Summary

- Working with variable sized arrays is more flexible with `vector`s in C++
- String operations are easier with C++ standard library

NPTEL