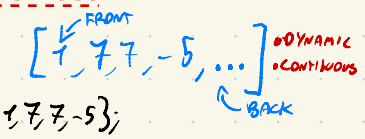
C++ Containers Docs

Doc VECTOR

#INCLUDE <VECTOR>



VECTOR<INT> V = {1,7,7,-5};
TYPE

V.INSERT(VECTOR.ITERATOR, VAL); | INSERTS VAL BEFORE ITERATOR POS. RETURNS ITERATOR TO INSERTED VALUE
V.POP_BACK();
V.PUSH_BACK(VAL);
V.CLEAR();
V.SIZE(); | N. OF ELEMENTS
V.EMPTY(); | BOOL TRUE IF EMPTY
VECTOR<TYPE>::CONST_ITERATOR IT = ...
V.BEGIN(); | VECTOR<T>::ITERATOR
V.END();
V.AT(INT POS) | REFERENCE TO ELEMENT CAN BE USED TO ASSIGN V.AT(5)=72; / V[INT POS] | REF. TO ELEMENT
V.FRONT() | REF. TO ELEMENT
V.BACK()
V.ERASE(ITR) || V.ERASE(ITR_FIRST, ITR_LAST) | REMOVE ELEMENT/S

Docs DOUBLE-ENDED QUEUE

#INCLUDE <DEQUE> {1,-7,5,...} • NON CONTINUOUS

DEQUE<INT> D = {7,5,-1,5};

SAME AS VECTOR

IN ADDITION:

D.PUSH_FRONT(VAL)
D.POP_FRONT()

COPY ARRAY

STD::COPY(START_PTR, END_PTR, DESTINATION);

Docs ALGORITHM

#INCLUDE <ALGORITHM>

SORT(ITR_BEGIN, ITR_END); | SORT < 0 1 2 3
SORT(ITR_BEGIN, ITR_END, GREATER<INT>()); | SORT > 3 2 1
BINARY_SEARCH(ITR_BEGIN, ITR_END, VALUE); | TRUE IF FOUND
REVERSE(ITR_BEGIN, ITR_END)
SHUFFLE (LOOK DOCS)

#INCLUDE <STRING> STRING METHODS

stoi (C++11), stol (C++11), stoll (C++11)	converts a string to a signed integer (function)
stoul (C++11), stoull (C++11)	converts a string to an unsigned integer (function)
stof (C++11), stod (C++11), stold (C++11)	converts a string to a floating point value (function)
to_string (C++11)	converts an integral or floating-point value to string (function)

S.FIND("CIAO") | AUTO POS = ... POSITION OF SUBSTRING NPOS IF NOT FOUND

CONCAT STRING AND CHAR

"A" + STD::STRING(1,'C');
OR
RESULT += "A";
RESULT += 'C';

STRING S;
S.NPOS;

READ LINE WITH SPACES

STD::STRING INPUT; STD::STRING TEMP_INPUT = "";
WHILE(STD::GETLINE(STD::CIN, TEMP_INPUT)){
INPUT += (TEMP_INPUT + '\N');
}

Docs STACK

#INCLUDE <STACK>

STACK<INT> S;

ELEMENT ACCESS
S.TOP(); | REF. TO TOP ELEMENT (LAST ELEM. PUSHED)

CAPACITY
S.EMPTY(); | TRUE IF EMPTY
S.SIZE(); | # OF ELEMENTS

MODIFIERS
S.PUSH(VAL); | INSERT AT TOP NO RETURN VAL.
S.POP(); | REMOVE TOP ELEMENT NO RETURN VAL.

Docs QUEUE

#INCLUDE <QUEUE>

QUEUE<INT> Q;

ELEMENT ACCESS
Q.FRONT(); | REF. FIRST ELEMENT (FIRST TO BE POPPED)
Q.BACK(); | REF. LAST ELEMENT (MOST RECENT PUSH)

CAPACITY
Q.EMPTY(); | TRUE IF EMPTY
Q.SIZE(); | # OF ELEMENTS

MODIFIERS
Q.PUSH(VAL); | INSERT AT END/BACK NO RETURN VAL.
Q.POP(); | REMOVE FIRST/FRONT ELEMENT NO RETURN VAL.

PRIORITY_QUEUE

UNDER #INCLUDE QUEUE

AT EVERY PUSH IT WILL REORDER

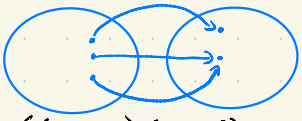
.POP()
.TOP() | WILL BE THE LARGEST ELEMENT
.PUSH()
.EMPTY()
.SIZE()

MIN PRIORITY

PRIORITY_QUEUE<INT, VECTOR<INT>, GREATER<INT>> MINQ;

Doc MAP

#INCLUDE <MAP>



• KEY_VALUE PAIRS
• SORTED BY KEYS
• KEYS ARE UNIQUE

MAP<INT, STRING> M = {{5,"HI"},{2,"P"}};
KEY VALUE

AUTO I = M.INSERT({5,"CIAO"}); | ITERATOR: I.FIRST → POINTS TO ELEMENT; I.SECOND (BOOL) → TRUE IF INSERTED, FALSE IF NOT (KEY ALREADY PRESENT)
INSERT IF NOT PRESENT — KEY VALUE PAIR

AUTO I = M.INSERT_OR_ASSIGN({KEY, VALUE}); | ITERATOR: I.FIRST → POINTS TO ELEMENT; I.SECOND (BOOL) → TRUE IF INSERTED, FALSE IF ASSIGNED
INSERT IF NOT PRESENT OR ASSIGN VALUE IF PRESENT

M[KEY] = VALUE
GET, UPDATE OR INSERT VALUE

AUTO ITR = M.FIND(5); | FINDS ELEMENT WITH KEY → ITERATOR TO VALUE; → END() IF NOT FOUND
M.SIZE(); | # OF ELEMENTS
M.EMPTY(); | TRUE IF EMPTY
M.CLEAR(); | CLEARS THE ELEMENTS
M.CONTAINS(KEY); | TRUE IF PRESENT, FALSE IF NOT
M.ERASE(ITR) || V.ERASE(ITR_FIRST, ITR_LAST) | REMOVE ELEMENT/S — FIRST INCLUDED, LAST EXCLUDED

Doc SET

#INCLUDE SET {1,-7,3,...} • UNIQUE VALUES • SORTED BY VALUES

SET<INT> S = {-7,-7,5,0}

AUTO I = S.INSERT(VALUE); | I.FIRST (ITERATOR) → POINTS TO ELEMENT; I.SECOND (BOOL) → TRUE IF INSERTED, FALSE IF NOT (VALUE ALREADY PRESENT)

AUTO ITR = S.FIND(VALUE) | FIND BY VALUE → ITERATOR TO VALUE; → END() IF NOT FOUND

S.SIZE(); | # OF ELEMENTS
S.EMPTY(); | TRUE IF EMPTY
S.CLEAR(); | CLEARS THE ELEMENTS
S.CONTAINS(KEY); | TRUE IF PRESENT, FALSE IF NOT
S.ERASE(ITERATOR); | REMOVE ELEMENT AT POS ITR # OF ELEMENTS DELETED

DEPO FILE
#INCLUDE <FORWARD>

PAIR

#INCLUDE <UTILITY>

AUTO P = MAKE_PAIR(EL1, EL2); — CAN BE DIFFERENT TYPES
PAIR<T1,T2>
P.FIRST | FIRST ELEMENT
P.SECOND | SECOND ELEMENT
GET<INDEX>(P);
GET<TYPE>(T); | FAILS IF TYPE NOT UNIQUE

TUPLE

#INCLUDE <TUPLE> — CAN BE DIFFERENT TYPES

AUTO T = MAKE_TUPLE(VAL1, VAL2, ...);
TUPLE<T1,T2,...>
GET<INDEX>(T); — YOU CAN THEN SET IT GET...<1> = VALUE;
GET<TYPE>(T); | FAILS IF TYPE NOT UNIQUE

VECTOR SNIPPETS

SHORT LOOP

```
FOR (AUTO E : V)
    COUT << E << ENDLN;
```

INPUT FROM CONSOLE

```
INT TEMP;
WHILE (CIN >> TEMP)
    V.PUSH_BACK(TEMP);
```

BAD MEMORY

```
TRY {
    V.PUSH_BACK(VAL);
} CATCH (CONST STD::BAD_ALLOC& E){
    STD::CERR << "MEM FAILED:" << E.WHAT() << ENDL;
}
```

Classes and Stuff

C -> C++

- To include C libraries
  - Just add C in front
    - `#include <cstdio>`
    - `#include <cassert>`
  - For external files/headers `#include "my_library.h"`
- NULL pointer is `int * p = nullptr;`
- Boolean type
  - `bool val = false`
    `val = true`

Namespaces

```
namespace usi {
  namespace math {
    int sum(...){...}
  } // math
} // usi

int main( ){
  ...
  int j = usi::math::sum(...);
}
```

I could have another namespace with another `sum` method

Skip the ::

```
using usi::math::sum;
```
or
```
using namespace usi::math
```
or
```
using namespace std
```

References

Alias to an existing object

```
int i = 7;
int &j = i;
j = 22;
```

Reference is not an object.

Reference to a const object

```
int sum(const int & a, const int & b) {
    return a + b;
}

int main() {
    int i = 7;
    int j = 2;
    std::cout << sum(i, j + 1) << std::endl;
}
```

Side Effects

```
void swap(int & a, int & b) {
    int tmp = a;
    a = b;
    b = tmp;
}

int main() {
    int i = 7;
    int j = 2;
    swap(i, j);
    std::cout << "i = " << i << std::endl << "j
}
```

Avoid Expensive Copy

```
void print_size_slow(vector<int> v) {
  std::cout << v.size() << std::endl;
}

void print_size_fast(vector<int> & v) {
  std::cout << v.size() << std::endl;
}

int main() {
    vector<int> A(1000);        // a vector of 1

    print_size_slow(A);         // makes a copy
    print_size_fast(A);         // operates dir
}
```

Return lvalues refs.

```
int M[2000];
int height = 40;
int width = 30;

int & m(int col, int row) {
  return M[col + row*width];
}

int main() {
    for (int i = 0; i < height; ++i)
        for (int j = 0; j < width; ++j)
            std::cin >> m(i,j); // stores a valu
    m(0,0) = 1;                 // stores a valu
}
```

Classes

Difference class - struct

Default access: Class : private
Struct : public

Class Example

```
class String {
public:
    // constructors (see below)
    String();
    String(const char *);
    String(const char * begin, const char * end);
    String(const char * begin, unsigned int len);
    String(const String &);

    // destructor
    ~String();

    // other, non-static member functions
    const char * c_string() const;
    unsigned int size() const;
    unsigned int append(const char *);

    // a static member function
    static unsigned int max_size();

private:
    // data members, all private
    char * data;
    unsigned int string_size;
    unsigned int allocated_size;
};
```

Extend a class

```
class Bis_String : public String {

};
```

Copy Constructor

```
class vec2d {
public:
    double x;
    double y;

    vec2d() : x(0), y(0) {};                    // "default" construct
    vec2d(double a, double b) : x(a), y(b) {};  // other constructor
    vec2d(const vec2d & v) : x(v.x), y(v.y) {}; // copy constructor
};
```

Constructors Example

```
class vec2d {
public:
    double x;
    double y;

    vec2d() : x(0), y(0) {
        std::cout << "default constructor." << std::endl;
    };

    vec2d(double a, double b) : x(a), y(b) {
        std::cout << "constructor with a=" << a << " b=" << b << std::endl;
    };

    vec2d(const vec2d & v) : x(v.x), y(v.y){
        std::cout << "copy constructor with v.x=" << v.x << " v.y=" << v.y << std
    };

    vec2d & operator = (const vec2d & v) {
        // (*this) is the implicit left-hand side operand of the assignment
        x = v.x;
        y = v.y;
        std::cout << "assignment with v.x=" << v.x << " v.y=" << v.y << std::endl;
        return *this;
    };

    vec2d operator - (const vec2d & v2) {   // Using the unary operator- with a single argument.
        return vec2d(x - v2.x, y - v2.y);   // vec2d result1 = a - b; // This would call a.operator-(b)
    }
};

std::ostream & operator << (std::ostream & output, const vec2d & v) {
    output << "(" << v.x << "," << v.y << ")";
    return output;
}

vec2d operator - (const vec2d & v) {
    return vec2d(-v.x, -v.y);
}                                           // -v
vec2d operator - (const vec2d & v1, const vec2d & v2) {   // evaluate at the definition
    return vec2d(v1.x-v2.x, v1.y-v2.y);                   // involve the class
}

bool orthogonal(vec2d u, vec2d v) { // u and v are also copy-constructed
    return (u.x*v.x + u.y*v.y == 0);
}

int main() {
    vec2d v1;                   // invokes default constructor
    vec2d v2(10, 3.2);          // invokes the second constructor
    vec2d v3(-2, 8);            // invokes the second constructor
    vec2d v4 = v2;              // invokes the copy constructor
    vec2d v5(v3);               // also invokes the copy constructor

    v1 = -v4;
    std::cout << "v1 = " << v1 << std::endl;
    std::cout << "v4 = " << v4 << std::endl;
    if (orthogonal(v1, v3))
        std::cout << "v1 and v3 are orthogonal." << std::endl;
    else
        std::cout << "v1 and v3 are not orthogonal." << std::endl;
};
```

Operator Overloading

- Before

```
void matrix_add(struct matrix * A, const struct matrix * B) {
```

- After

```
matrix & operator += (matrix & A, const matrix & B) {
    /* computes A += B and returns A */
    if (A.cols != B.cols || A.rows != B.rows)
        return A;
    int * a = A.data;
    const int * a = B.data;
    for (unsigned int i = 0; i < A.rows; ++i)
        for (unsigned int j = 0; j < A.cols; ++j)
            *a++ += *b++;
    return A;
}

int main() {
    struct matrix A;
    struct matrix B;

    /* ... */
    A += B;
}
```

Templates

Methods

```
template <typename T>
unsigned int array_size(unsigned int n) {
    return n * sizeof(T);
}
```

Usage

```
std::cout << array_size<char>(100) << endl;
```

Class

```
/* a single-link list of doubles */
template <typename T>
class list {
private:
    struct list_elem {
        T value;
        list_elem * next;
        list_elem(const T & v, list_elem * n)
            : value(v), next(n) {};
    };

    list_elem * first;

public:
    list()
        : first(nullptr) {};

    void list_add(const T & v) {
        first = new list_elem(v, first);
    }

    bool list_find(const T & v) {
        for (const list_elem * i = first; i != nullptr; i = i->next)
            if (i->value == v)
                return true;
        return false;
    }
};
```