C++

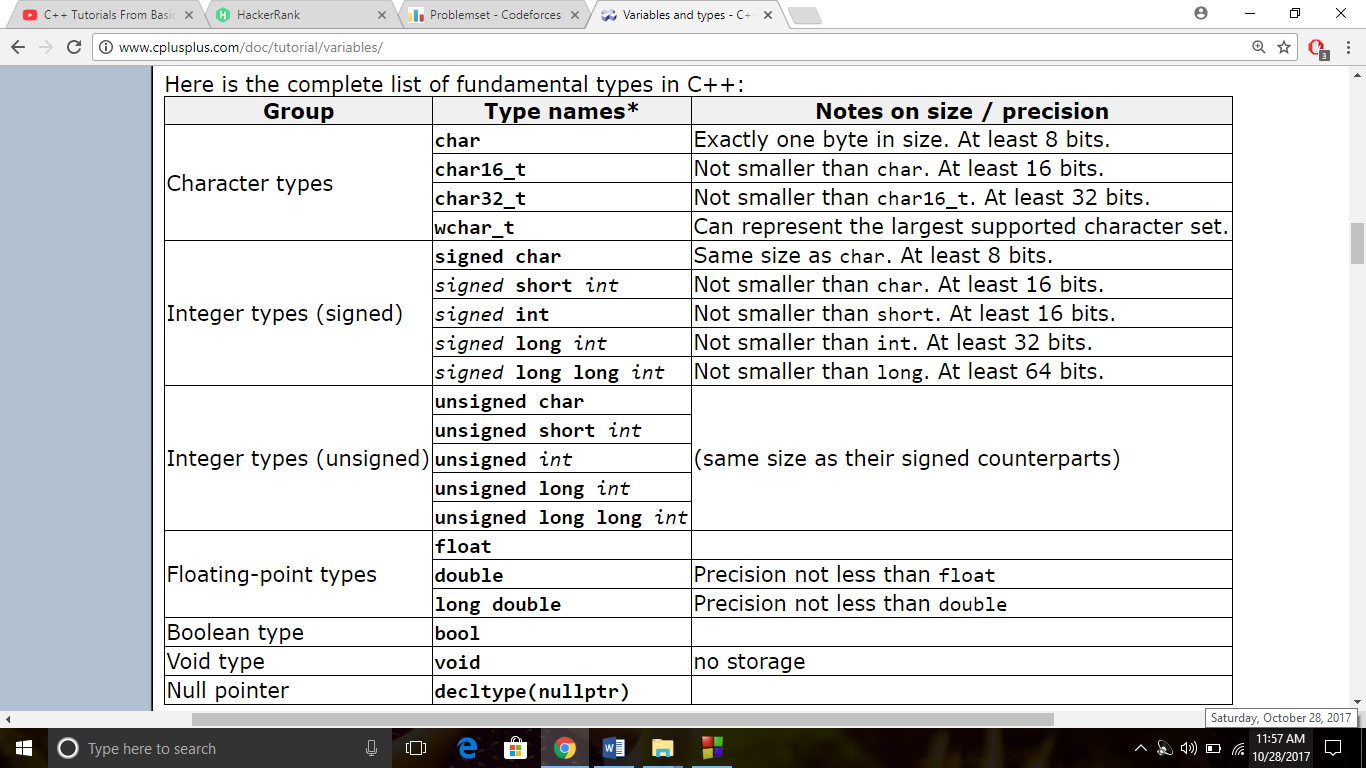
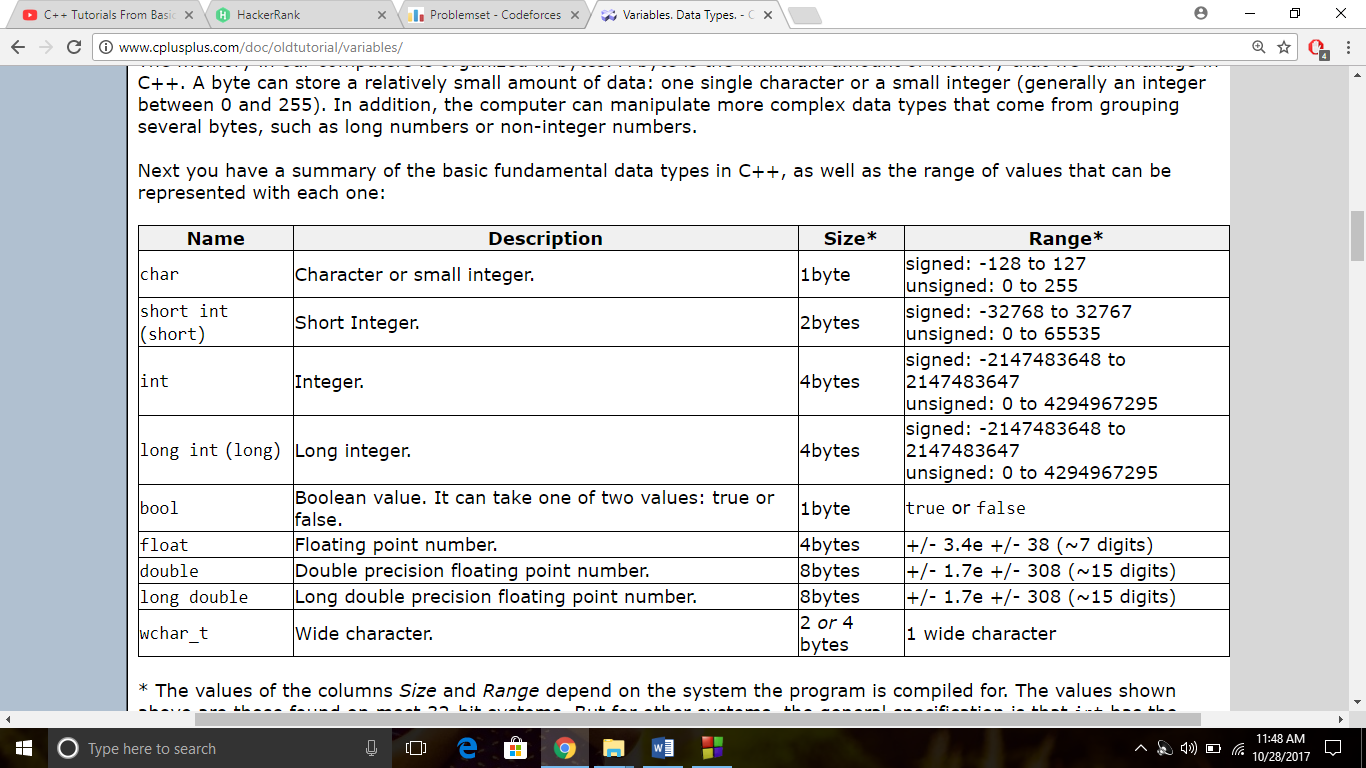
\*\*\***#include <bits/stdc++.h>** --> Used to include all (most of ) the standard library header files.

1s 🡪 10^7

2s 🡪 10^7

4s 🡪 10^8

Different Data types





\*\*\* Use long long int if the integer has value greater than (approximately 9 digits or ) 2\*109).

i.e., while you int or long make sure that the value is less than 109 .

*Formatting data*

\*\*\*To use the below things include <iomanip>

1)setw()

2)setpresicion

3)fixed

4)showpoint

To set precision to a decimal such as upto a certain no. of decimal places use the following format.

std::printf("%d \n%ld \n%c \n%3f \n%9lf",d,ld,c,f,lf);

↘Names of the variables

Cout

1) cout << ++x <<endl;

implies 1 is added to x and then the number is printed

For cout << x++ << endl;

x is printed first and then 1 is added to it

So, if we type

Eg: int x= 20;

cout<<x++<<endl;

cout << x <<endl;

the output will be 20

21

whereas if we type

int x= 20;

cout<<++x<<endl;

cout << x <<endl;

The output will be 21

21

2) x+=2; implies x=x+2;

lly, x\*=2 implies x=x\*2;

--> int a = int('c');

(OR) int a = 'c';

(OR) int a= char('c');

(OR) int a= (char)'c';

(OR) int a= (int)'c';

var. a will be equalised to the integer equal to ASCII code of character entered, here 'c'.

--> char ch = char(99);

(OR) char ch = int(51);

(OR) char ch = 51;

(OR) char ch =(int)99;

(OR) char ch = (char)99;

var. ch will be equlised to the character whose ASCII code will be the interger.

* But while writing a statement like

cout << int('c')<<" "<<int(99)<<" "<<char('c')<<" "<<char(99);

The output will be 99 99 c c

BUT NOT for {char ch ='99';}

TYPE CASTING

--> int a = 1025;

int \*p = &a;

char \*p0;

p0 = (char\*)p;

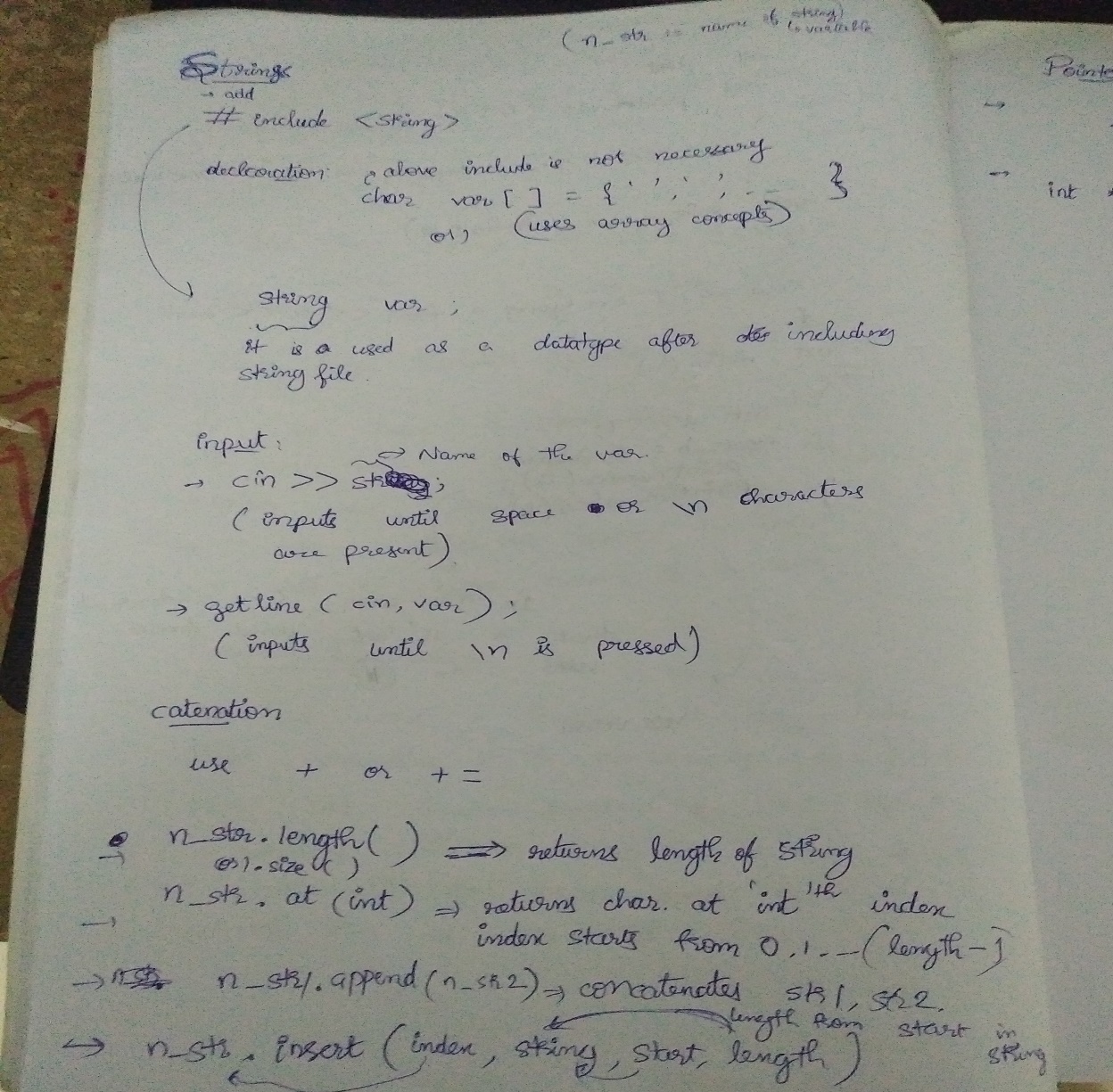
cout<< "The address of a is "<<

*Conversion of String to Integer*

* Use stoi(name\_string or “15672”) //returns the integer in the string or 15672.

*Conversion of Int to String*

* Use std::to\_string(name\_int) //returns the data type string.

**String functions**

|  |  |
| --- | --- |
| // inserting into a string  #include <iostream>  #include <string>  int main ()  {  std::string str="to be question";  std::string str2="the ";  std::string str3="or not to be";  std::string::iterator it;  // used in the same order as described above:  str.insert(6,str2); // to be (the )question  str.insert(6,str3,3,4); // to be (not )the question  str.insert(10,"that is cool",8); // to be not (that is )the question  str.insert(10,"to be "); // to be not (to be )that is the question  str.insert(15,1,':'); // to be not to be(:) that is the question  it = str.insert(str.begin()+5,','); // to be(,) not to be: that is the question  str.insert (str.end(),3,'.'); // to be, not to be: that is the question(...)  str.insert (it+2,str3.begin(),str3.begin()+3); // (or )  std::cout << str << '\n';  return 0;} |  |

string::erase

#include <iostream> /

#include <string> \*\*\*--/

int main ()

{

std::string str ("This is an example sentence.");

std::cout << str << '\n';

// "This is an example sentence."

// ^^^^^^^^

1 /🡪 Starting index. |

/ /🡪 no. of chars |

/ / |

str.erase (10,8); 🡨|

std::cout << str << '\n';

2 Location of start 🡨\ /🡪 index of char that is to be removed.

(Constant for all) \ / // "This is an sentence."

str.erase (str.begin()+9); // ^

std::cout << str << '\n';

// "This is a sentence."

3 str.erase (str.begin()+5, str.end()-9); // ^^^^^

std::cout << str << '\n'; ↘🡪 //first is included (to delete) and second is excluded.

// "This sentence."

return 0;

}

Substring

String s1=”Hello, This is Psp. Hai Guys!!”,s2;

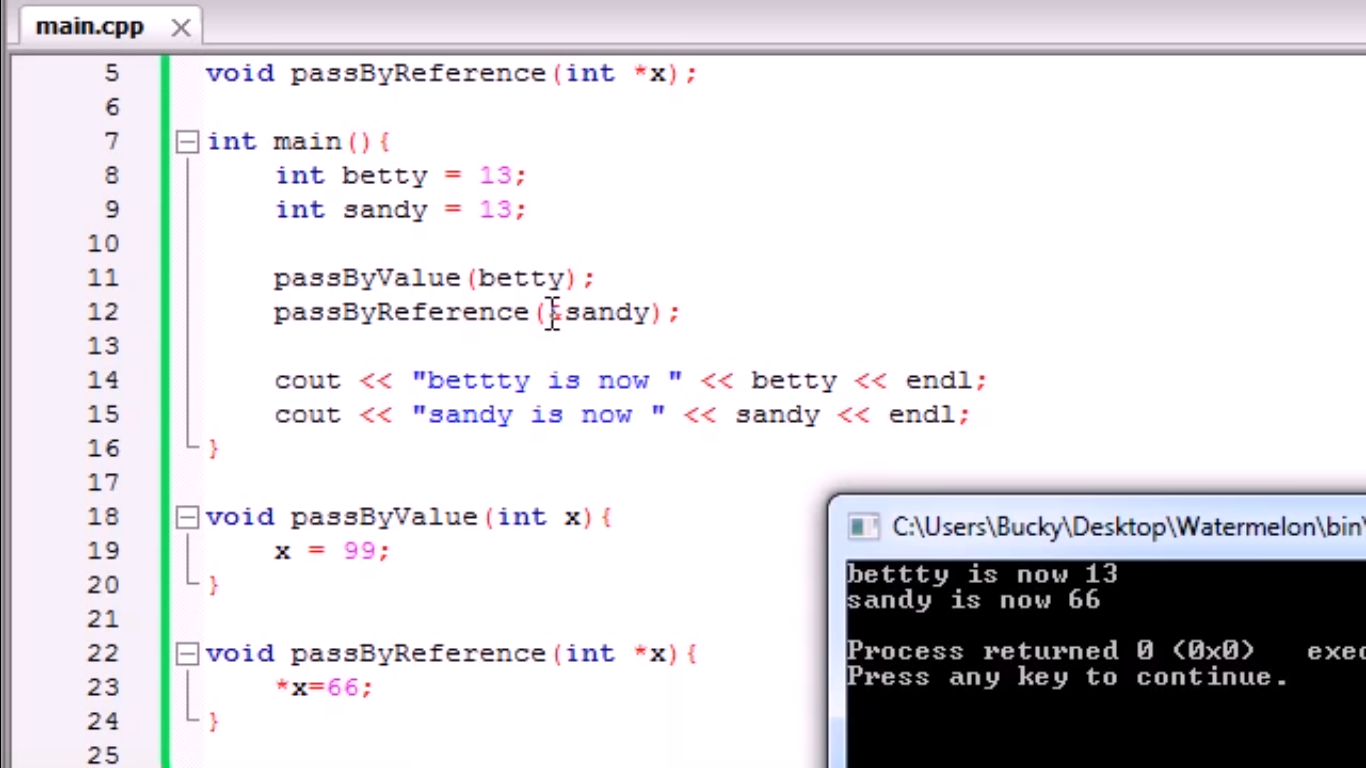
s2.assign(s1,7,4); // s2 will be equal to “This”.

(Or)

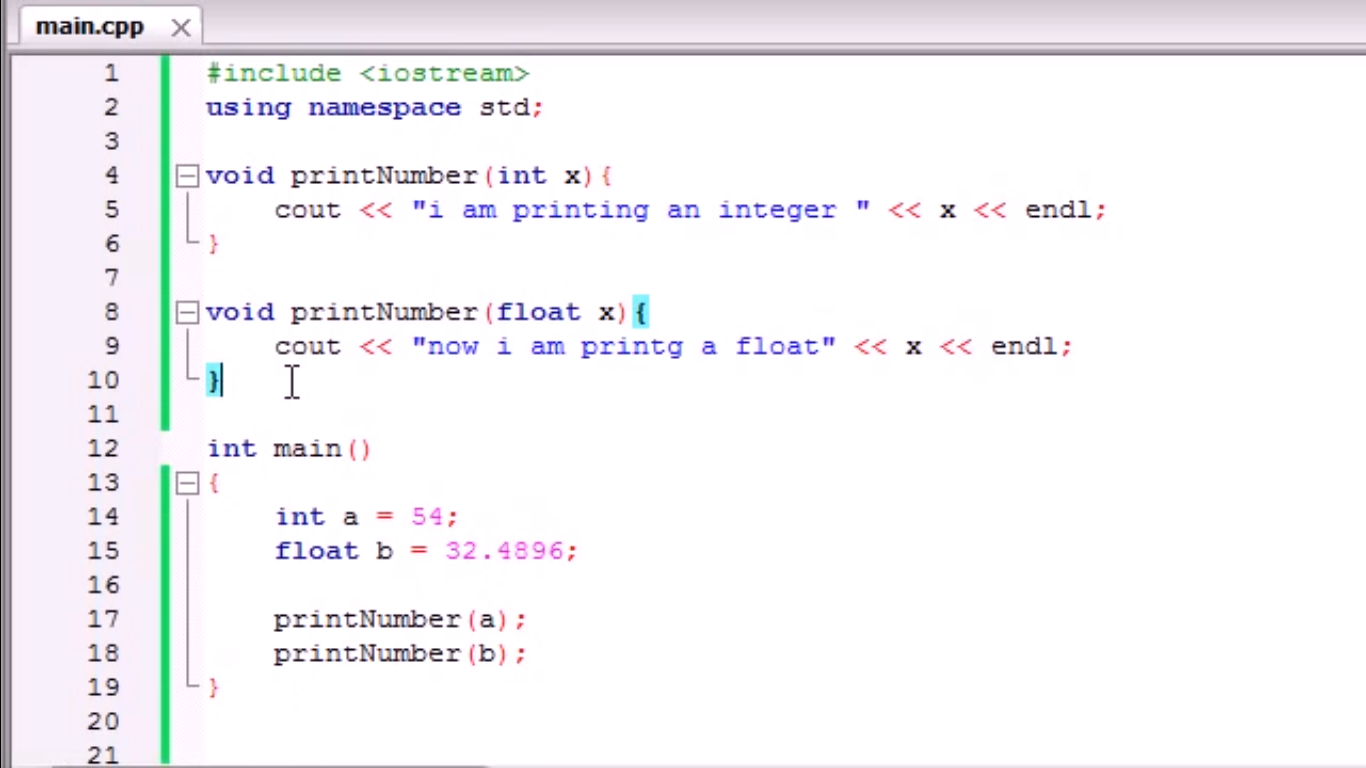
S2 = s1.substr(7,4)

***Pointers***

***Pass by value and pass by reference***



**Function Overlaoding**



***Vectors and Vector functions***

--> //A vector need not be sized initially

vector <int> v; // or vector <int> v(n);

--> v.push\_back(10);// Adds number 10 at the end

v.push\_back(12);

v.push\_back(15);

v.push\_back(4);

* v.pop\_back() removes the last element of the vector.

for(int i=0;i<v.size();i++)

cout<<v[i]<<" ";

cout<<endl;

-->//v.size() returns the size of the vector

-->//v.resize(n) resizes the vector to size n

--> v.insert(v.begin()+2, x );//Adds element x BEFORE the element at specified index

for(int i=0;i<v.size();i++)

cout<<v[i]<<" ";

cout<<endl;

-->v.erase(v.begin()+3);//Removes element at specified index

\*\*\*//v.begin(),v.end() is used to give the location of 1st and last elements of the vectors.(Its an iterator slightly diff. from pointer )

// whereas v.front(),v.back() is used to give the 1st and last element of the vectors.

for(int i=0;i<v.size();i++)

cout<<v[i]<<" ";

cout<<endl;

--> //v.empty() returns a Boolean value whether the vector is empty or not

--> v.clear();

//This is clearing which implies deleting each element \*not assigning it to zero\*.

if(v.empty())

cout<<"empty";

else

cout<<"not empty";

***(1) empty container constructor (default constructor)***

Constructs an [empty](http://www.cplusplus.com/vector::empty) container, with no elements.

***\*\*(2) fill constructor***

Constructs a container with *n* elements. Each element is a copy of *val*.

***(3) range constructor***

Constructs a container with as many elements as the range [first,last), with each element constructed from its corresponding element in that range, in the same order.

***(4) copy constructor***

Constructs a container with a copy of each of the elements in *x*, in the same order.

// constructing vectors

#include <iostream>

#include <vector>

int main ()

{

// constructors used in the same order as described above:

1) std::vector<int> first; // empty vector of ints

2) std::vector<int> second (4,100); // four ints with \*\*\* value 100

3) std::vector<int> third (second.begin()+1,second.end()-1); // iterating \*\*\* through second

4) std::vector<int> fourth (third); // a copy of third

// the iterator constructor can also be used to construct from arrays:

int myints[] = {16,2,77,29};

3 std::vector<int> fifth (myints, myints + sizeof(myints) / sizeof(int) );

\

↘🡪 no. of elements

//(or)vector <int> fifth(myints + i , myints + k); elements [i,k) of myints will constructed to “fifth” vector.

//which implies that just writing n\_array or n\_array + i treats them as iterators.

(my interpretation)

return 0;

}

Operator ‘=’

vector <int> v1(4,0);

vector <int> v2(7,3);

v2 = v1; // Assigns v1 to v2 erasing all the data initially present in v2.

v2 = vector <int> (); // will assign v2 to an empty vector. Replace (..) by any construction to assign anything to v2.

↘🡪 Bracket is to be given unlike declaration statement.

↘🡪 Note: Name is not written unlike declaration statement.

\*\* The above line can’t be used to initialize v2 i.e., it has to be initialized before.

v1.swap(v2); // swaps v1 & v2; irrespective of initial sizes.

Functions from Algorithms file

The most useful functions are available @: https://www.geeksforgeeks.org/c-magicians-stl-algorithms/

* **accumulate(first\_iterator, last\_iterator, initial value of sum)** – Does the summation of vector elements
* **reverse(first\_iterator, last\_iterator)** – To reverse a vector.
* **count(first\_iterator, last\_iterator,x)** – To count the occurrences of x in vector.
* **find(first\_iterator, last\_iterator, x)** – Points to last address of vector ((name\_of\_vector).end()) if element is
* [**binary\_search**](http://quiz.geeksforgeeks.org/binary-search-algorithms-the-c-standard-template-library-stl/)**(first\_iterator, last\_iterator, x)** – Tests whether x exists in sorted vector or not.

// boolean return type fn, finds x in b/w the given [first,last) iterators and returns 1 if element is present in the (vector or kinda) else 0.

\*// works only when elements b/w iterators are sorted.

* **lower\_bound(first\_iterator, last\_iterator, x)** – returns an iterator pointing to the first element in the range [first,last) which has a value not less than ‘x’. (>=)
* **upper\_bound(first\_iterator, last\_iterator, x)** – returns an iterator pointing to the first element in the range [first,last) which has a value greater than ‘x’. (>)
* **arr.erase(unique(arr.begin(),arr.end()),arr.end())** – This erases the duplicate occurrences in sorted vector in a single line. \*\*\*
* **next\_permutation(first\_iterator, last\_iterator)**– This modified the vector to its next permutation.
* **prev\_permutation(first\_iterator, last\_iterator)**– This modified the vector to its previous permutation.

<algorithm> has to be included (if bits/stdc++.h is not)

swap(x,y) swaps two vectors (irrespective of initial sizes )or variables but not arrays(as a whole);

Swap\_ranges

swap\_ranges(v1.begin()+i,v1.begin()+j, v2.begin() +k {or n\_array} )

swaps[i,j) th index elements of v1 with elements starting from v2.begin() +k ;

replace (myvector.begin(), myvector.end(), 20, 99);

//replaces all the numbers that are = 20 by 99 in the above range of the vector myvector.

partial\_sort (myvector.begin(), myvector.begin()+5, myvector.end());

\ \ ↘🡪 ending position

\ ↘🡪starting position

↘🡪Position to place the sorted elements.

\*min\_element(v.begin(),v.begin+k); // returns min element in the given range.

\*max\_element(v.begin(),v.begin+k); // returns max element in the given range.

min\_element(v.begin(),v.begin+k); // reprensts the min element’s iterator (i.e., location can be used to refer but can’t be printed ) in the given range.

||y, for max\_element(… , …);

If there are many possible min/max values for the it gives the first occurred one.

// iter\_swap example

int myints[]={10,20,30,40,50 }; // myints: 10 20 30 40 50

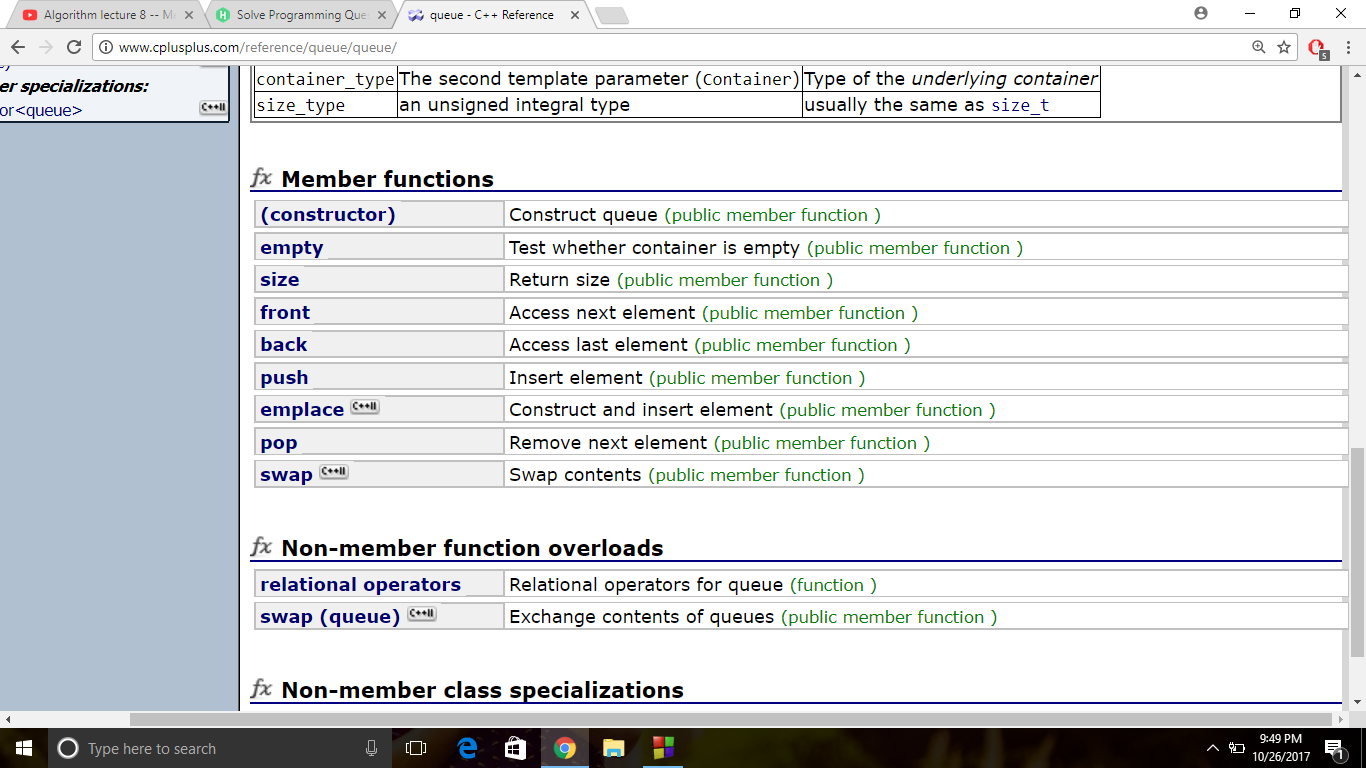
std::vector<int> myvector (4,99); // myvector: 99 99 99 99

std::iter\_swap(myints,myvector.begin()); // myints: [99] 20 30 40 50

// myvector: [10] 99 99 99

std::iter\_swap(myints+3,myvector.begin()+2); // myints: 99 20 30 [99] 50

// myvector: 10 99 [40] 99



Templates

QUEUE

A Queue is only used when the first element is needed.

Let q be the name of queue.

Initialising - queue <data\_type> q;

int x;

q.push(x); --> Adds x at the end of the queue.

q.pop(); --> pops out the first element

q. front() --> Refers to first element

STACK

A stack is used to access the top element

stack <data\_type> s;

s.push(x) --> Adds x at the end of the queue.

s.pop() --> Removes the last element of the stack.

s.top() --> Refers to the last element in the stack.

NOTE : There is no top fn. in queue and no front fn. in stack.

ITERATOR

vector <int>::iterator it; --> Initialising a vector // "vector" can be replaced by any other data structure. , "it" is the name of the iterator

NOTE : Iterator is a data structure that works like pointer but not exactly a pointer so \*it is used to refer to the element at that location.

An iterator can’t be printed out it is only used for referring elements whereas pointers can be printed out.

/\* // **Usage of iterator - example**

int x;

vector <int> v(0);

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

{ cin>>x;

v.push\_back(x);

}

vector <int>::iterator it;

for(it=v.begin();

it!=v.end();it++)

{

cout<<\*it<<endl;

}

\*/

Pair

pair<int,int> pr;

pr=make\_pair(x,y);

**Creating a vector of pairs**

#include <bits/stdc++.h>

using namespace std;

int main() {

int x,i=0;

char y; / /-> necessary space so that it will not become ‘>>’ operator (cin)

vector <pair<int,char> > :: iterator it;

vector <pair<int,char> > v(26);

pair<int,char> pr;

for(it=v.begin();it<v.end();it++)

{

\*it = (make\_pair(i+1,'a'+i));

i++;

}

for(it=v.begin();it<v.end();it++)

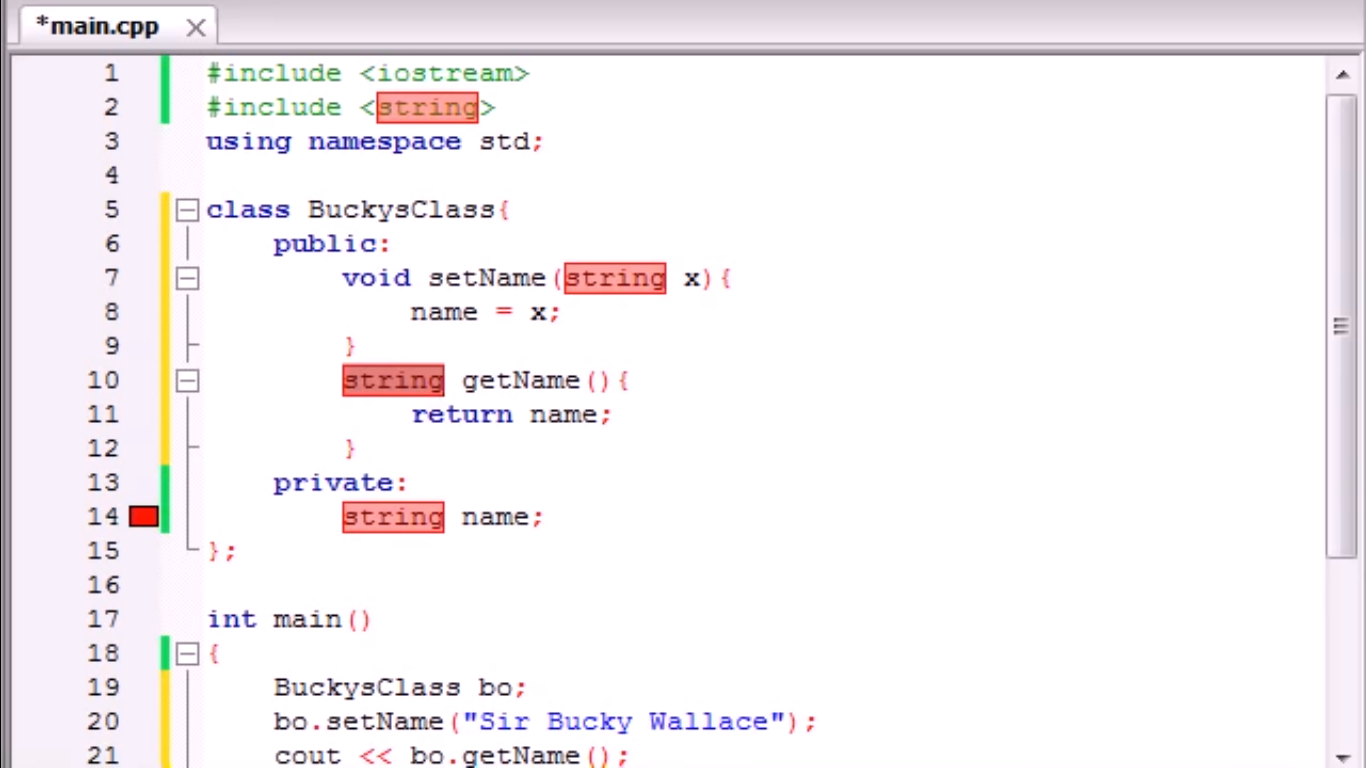
{

cout<<(\*it).first<<" "<<(\*it).second<<endl;

}

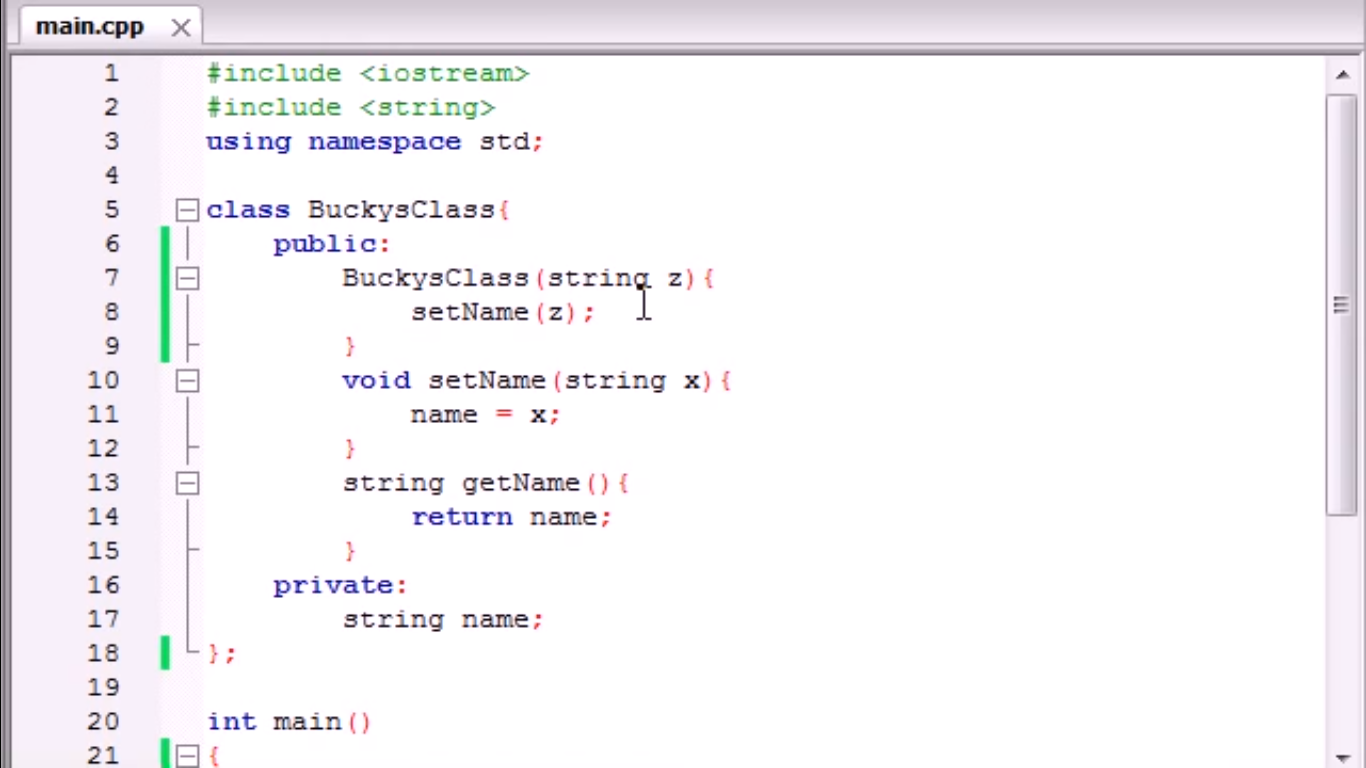
return 0;

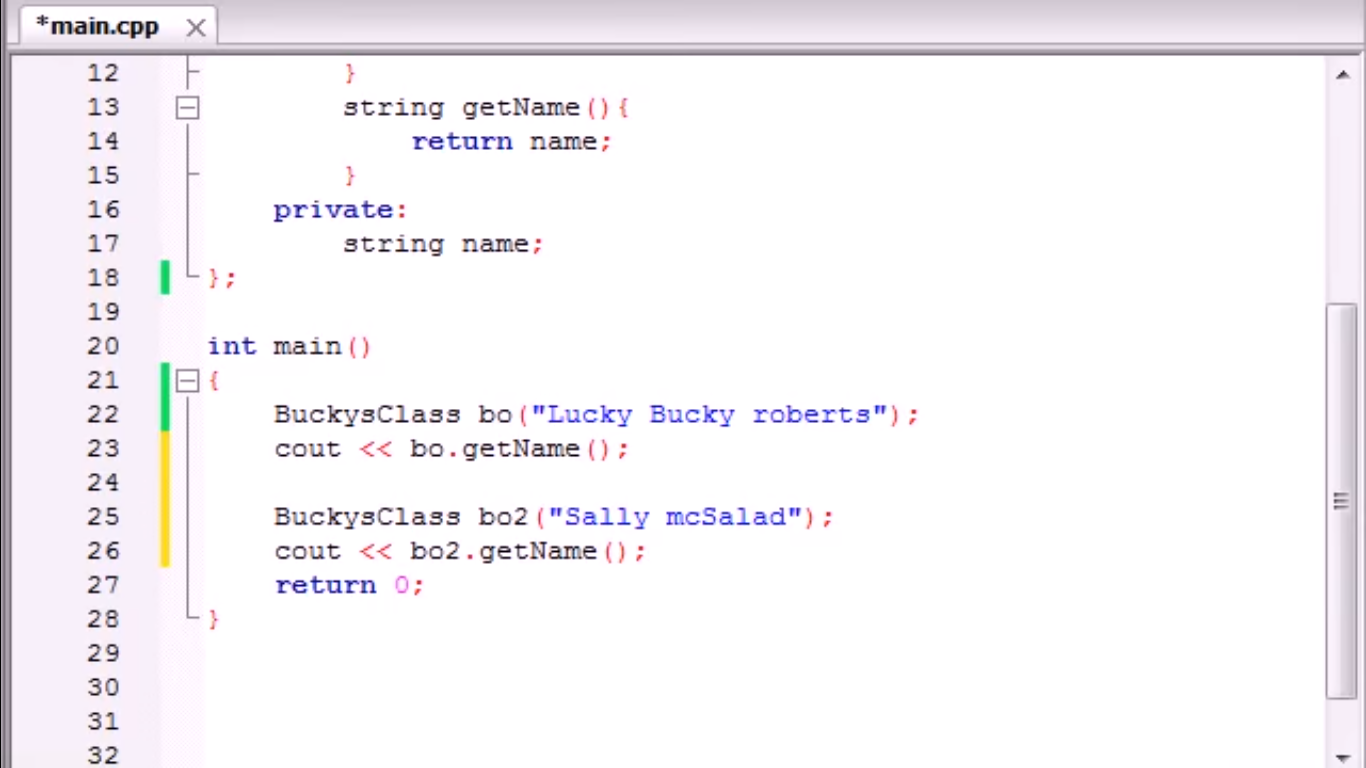
}

***Classes and public, private***

Constructors

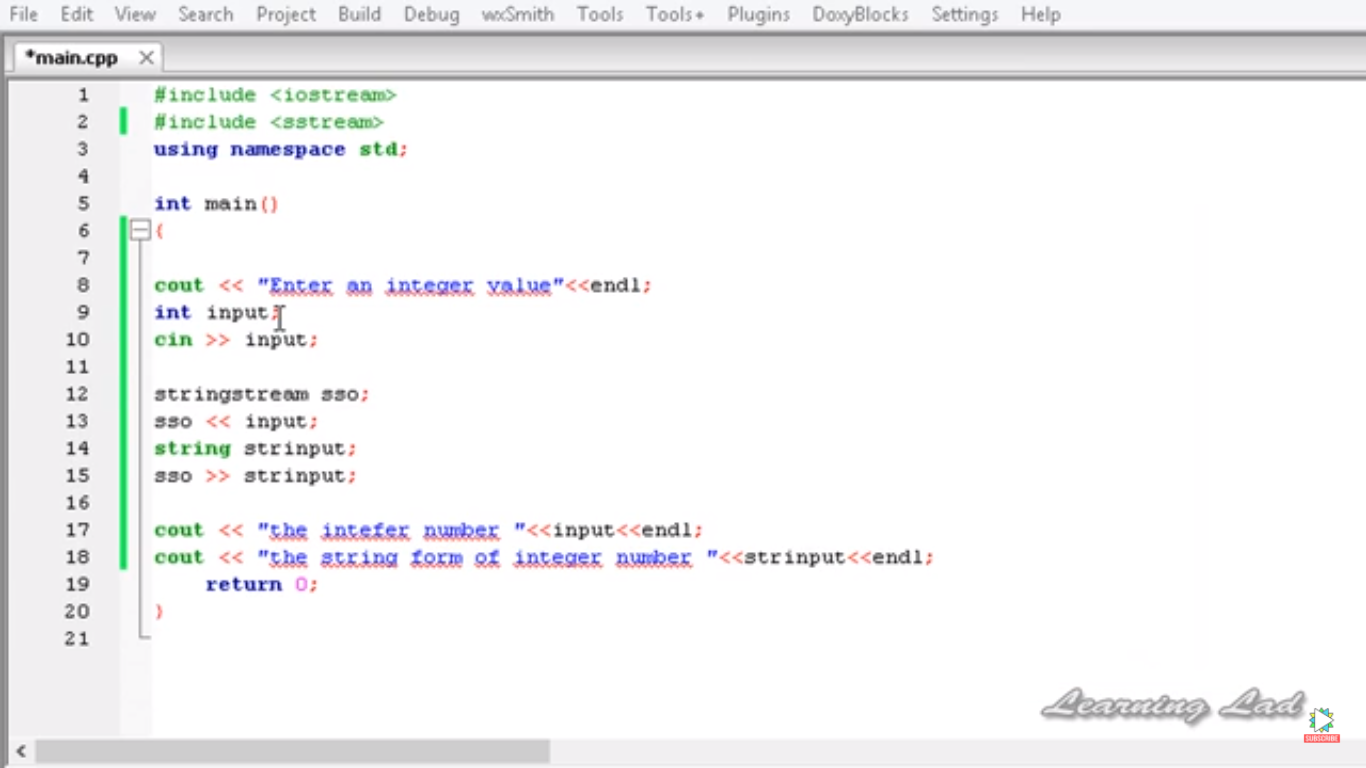
* Constructor is something like function that gets called automatically when a class is called.
* It is generally used to initialize variables in a class.
* It is created by making a function with the same name as the name of class in which it is in.



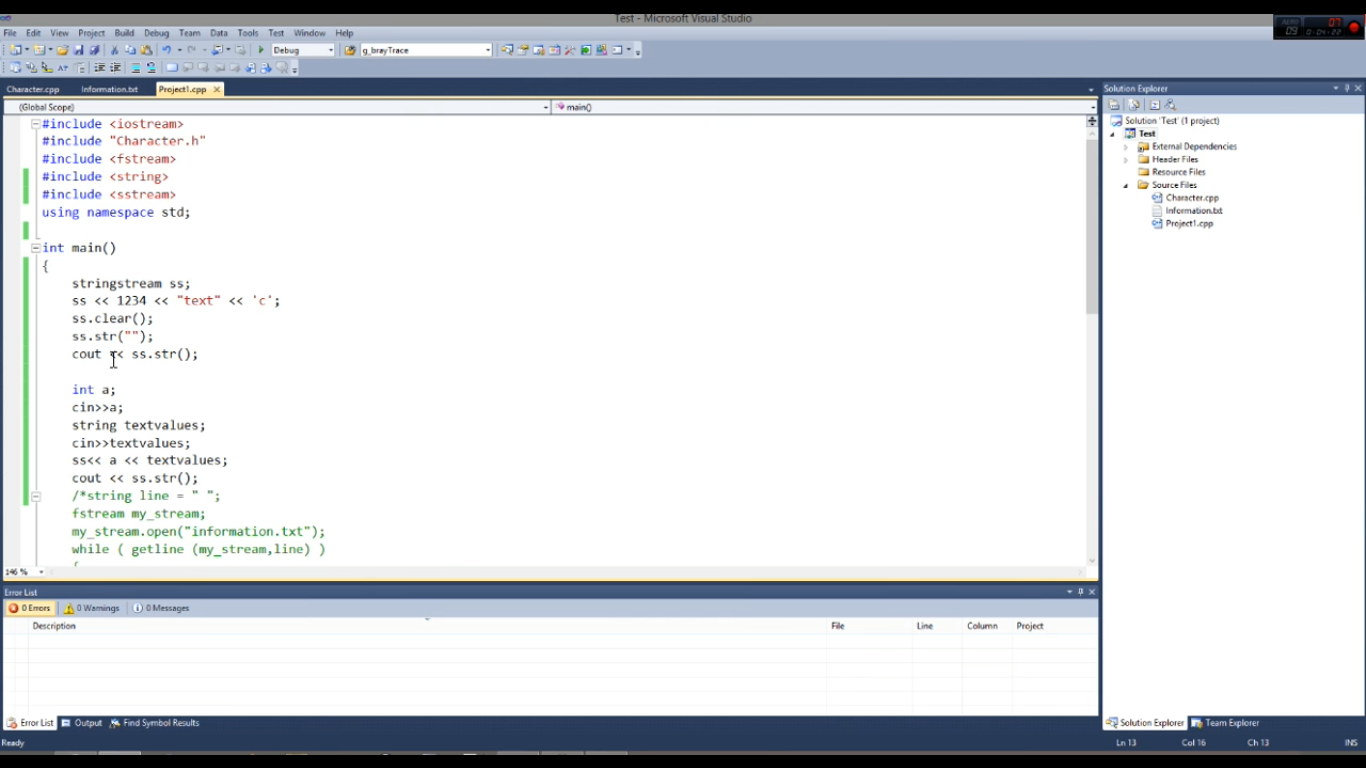


bo and bo2 are two objects of the above class.

The output is Lucky Bucky robertsSally mcSalad i.e., both the above names but the variable name is not overwritten which means that bo.getName() is one function and bo2.getName() is another function i.e., multiple variables can be created with same name using different objects in a class.

**String Stream**

Converting different data types to a single string.



* String stream is used to convert int to text and text to int.

(31/10/17)

Maps

map <data\_type1,data\_type2> m;

//Let datatype1 be char , datatype2 be int.

use m[‘a’] = 3; // To map char. ‘a’ --> int 3; --> The statement used for initialisation also;

m['a'] = 5; // m[‘a’] second time to refer the mapped element or change its value.

\

\ --> The element used to refer the other element.

Note : Size of map is not declared initially and it is added everytime with the above syntax.

Direct Sort Function

sort(a,a+n) ; --> To sort a array whose size is n

sort(v.begin(),v.end()); To sort a vector

(OR) sort(&a[0],&a[n]) , n is no. of elements in a vector == a.size().

sort(loc./iterator Of postion1, loc./iterator Of postion2) sorts the elements in middle(1st included & 2nd isn’t ).

Note : sort function may be used to sort any data structure.

Direct Search Functions

\*binary\_search(first,last,x)

Sdf

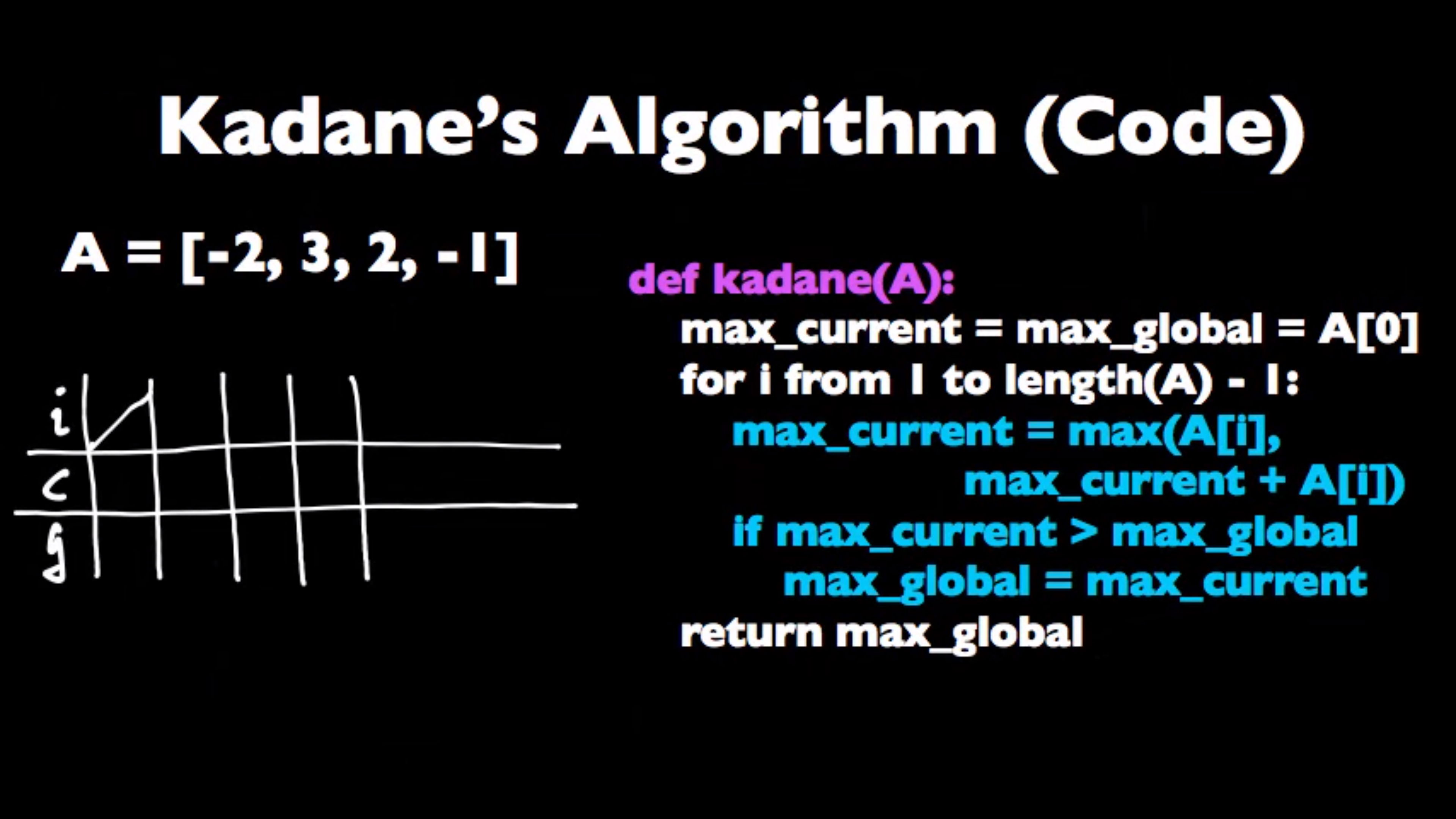
Sd

Fsfd

\*\*\*for(auto n\_var : n\_vector) works like

for i in a : #as in python

Kadane’s Algorithm to find Max. Sub Array Sum



**Data Types**

**Abstract data type :**

**Static Data Type :**

**Dynamic Data Type :**

**Linked Lists**

**Singly Linked List :**