23/5

#include<iostream>

#include<list>

using namespace std;

// individual element of list is called --> Node.

int main() {

list <int> Arru; // no size declaration as dynamic memory

Arru.push\_back(20); // inserting in the end

Arru.push\_back(30);

Arru.push\_front(10); // inserting in the front

Arru.push\_back(100);

// for deleting -->

Arru.erase(Arru.end()); // Note that all these arent values..they're addresses. Arru.begin() or Arru.end() are addresses

/\*

for output of list, we go through each of the nodes individually

this is done by a for loop, and instead of using an "int i" , we use an 'iterator' to iterate the list

the iterator is the data type, and it will have a name, here --> it

\*/

for (list<int>::iterator it = Arru.begin(); it != Arru.end(); it++) {

// scope resolution operator :: to show iterator is part of the list

// starts at beginning of the list, using the function .begin()

// stops when equal to last term of the list

// incremented by one value each time like usual

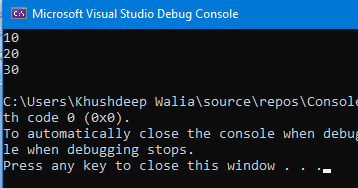
cout << \*it << endl; // VERY VERY IMP --> iterator points to the adress of the next element

// To get the value at the address, DEREFERENCE it using \* , like we do in pointers

}

return 0;

}



**IMPLEMENTING LISTS IRL BY MAKING BATTLE ROYALES MATCHMAKING:**

#include<iostream>

#include<list>

using namespace std;

void DisplayRating( list <int> a ) {

for (list<int> ::iterator it = a.begin(); it != a.end(); it++) {

cout << "rating: " << \*it << endl;

}

}

int main() {

// list of exp level of all players waiting in a lobby -->

list<int> AllPlayerEXP = { 2,9,6,7,3,1,4,8,3,2,9 };

// different lists for noob and pros -->

list<int> noobs; // Rated 1-5

list<int> pros; // Rated 6-10

for (list<int> ::iterator it = AllPlayerEXP.begin(); it != AllPlayerEXP.end(); it++)

{

int rating = \*it; // cause remember 'it' is the pointer, and '\*it' is the value towards which its pointing

if (rating <= 5 && rating >= 1)

noobs.push\_back(rating);

else if (rating >= 6 && rating <= 10)

pros.push\_back(rating);

}

// DisplayRating(noobs) << endl; // in case you forgot apparently you cant put endl after calling functions

cout << "RATING OF NOOBS: " << endl;

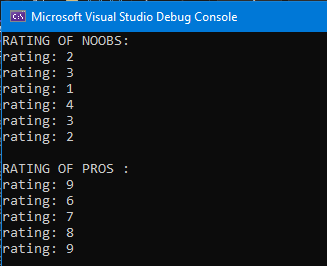
DisplayRating(noobs);

cout << "\nRATING OF PROS : " << endl;

DisplayRating(pros);

return 0;

}

****

**SUPPOSE U WANT THE RATINGS TO BE ORDERED…INCREASING**

**THEN CHANGE CODE LIKE THIS 🡪**

#include<iostream>

#include<list>

using namespace std;

void DisplayRating(list <int> a) {

for (list<int> ::iterator it = a.begin(); it != a.end(); it++) {

cout << "rating: " << \*it << endl;

}

}

ARMAAN YOU HAVE TO PASS LIST BY REFERENCE CAUSE VERNA IT WILL MAKE A COPY OF THE LIST AND WORK ON IT AND IT WONT CHANGE THE ORIGINAL LIST.

void InsertPlayerIntoOrderedList(int NewPlayerRating, list<int> &SortedList) {

for (list<int> ::iterator it = SortedList.begin(); it != SortedList.end(); it++) {

//

if (\*it > NewPlayerRating) {

SortedList.insert(it, NewPlayerRating); // 'it' is position, so we putting value of rating at that position (address,value) format

return;

}

}

SortedList.push\_back(NewPlayerRating); // if u find no element bigger than the ranking, add it in the back of the list

}

int main() {

// list of exp level of all players waiting in a lobby -->

list<int> AllPlayerEXP = { 2,9,6,7,3,1,4,8,3,2,9 };

// different lists for noob and pros -->

list<int> noobs; // Rated 1-5

list<int> pros; // Rated 6-10

for (list<int> ::iterator it = AllPlayerEXP.begin(); it != AllPlayerEXP.end(); it++)

{

int rating = \*it; // cause remember 'it' is the pointer, and '\*it' is the value towards which its pointing

if (rating <= 5 && rating >= 1)

InsertPlayerIntoOrderedList(rating, noobs);

else if (rating >= 6 && rating <= 10)

InsertPlayerIntoOrderedList(rating, pros);

}

// DisplayRating(noobs) << endl; // in case you forgot apparently you cant put endl after calling functions

cout << "RATING OF NOOBS: " << endl;

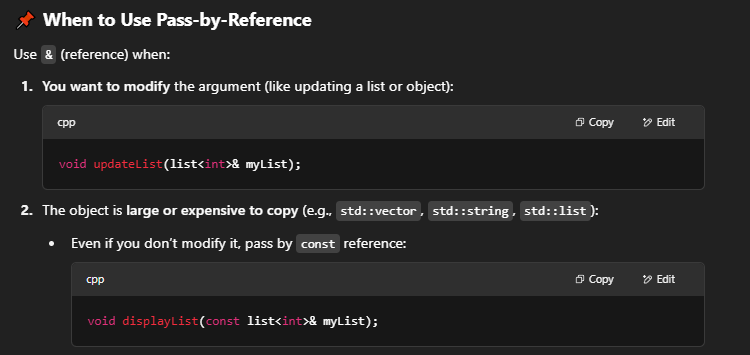
DisplayRating(noobs);

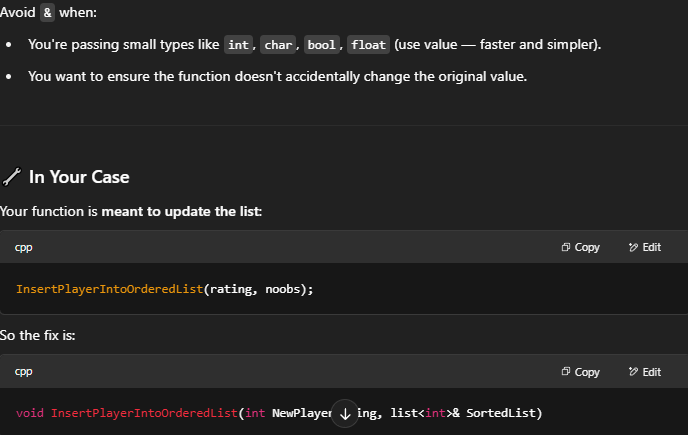
cout << "\nRATING OF PROS : " << endl;

DisplayRating(pros);

return 0;

}

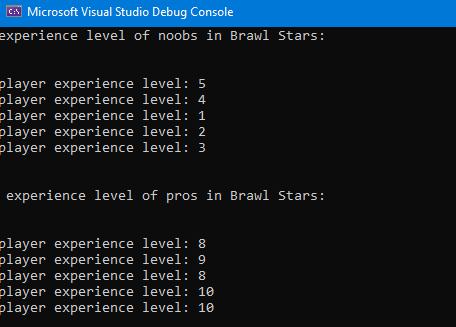
****

****

****

**24/5**

**LETS GOOO YAAD HAI !!!**

****

**25/5**

**input 🡪**

#include<iostream>

#include<list>

using namespace std;

template <typename arru>

class BaseClass {

public:

list<arru> l1;

void input(int cnt) {

cout << "enter " << cnt << " elements for the list: " << endl;

for (int i = 1; i <= cnt; i++) {

arru value;

cout << "enter value of " << i << " value: " << endl;

cin >> value;

l1.push\_back(value);

}

}

void show() {

cout << "following are the elements of the list: " << endl;

for (typename list<arru> ::iterator it = l1.begin(); it != l1.end(); it++) {

cout << \*it << endl;

}

}

};

int main() {

BaseClass<int> b1;

b1.input(10);

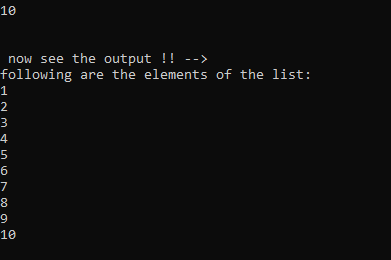
cout << "\n\n now see the output !! -->" << endl;

b1.show();

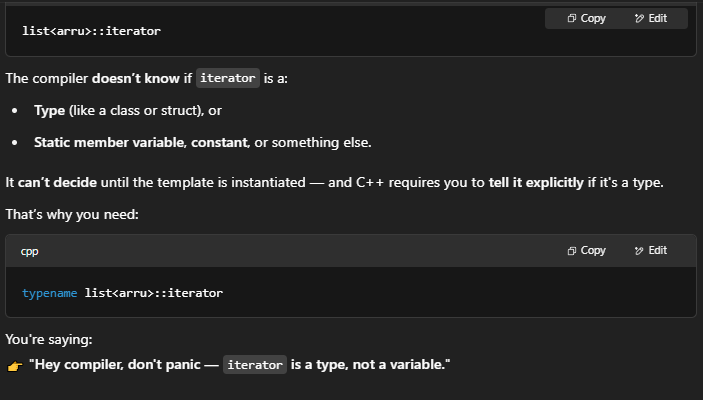
cout << "\n\n cool right ??";

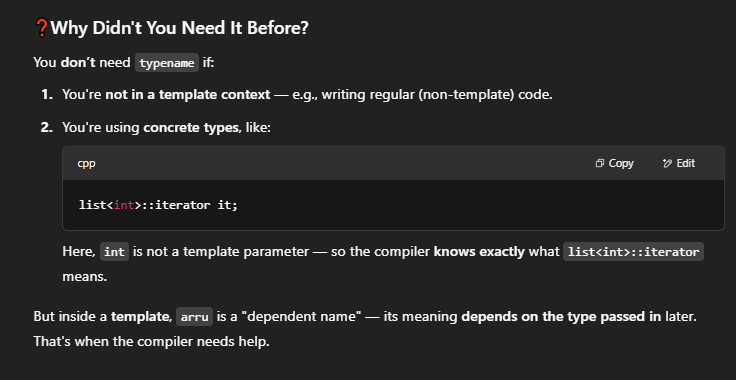
return 0;

}

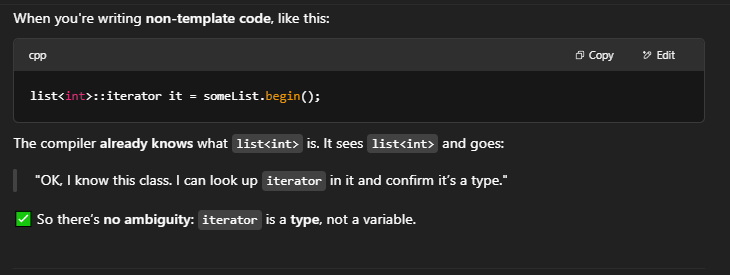
****

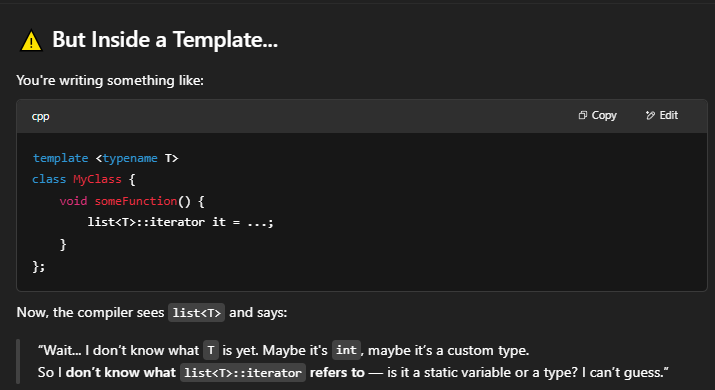
**SEE WHY YOU DIDN’T NEED ‘TYPENAME’ KEYWORD PEHLE ----🡪**

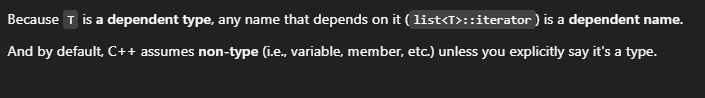
****

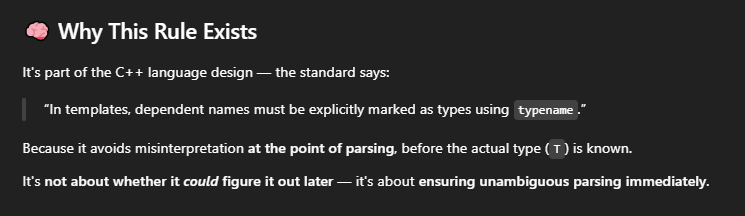
****

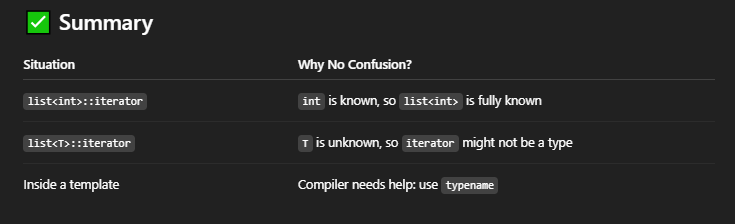
**FURTHER EXPLANTION 🡪**

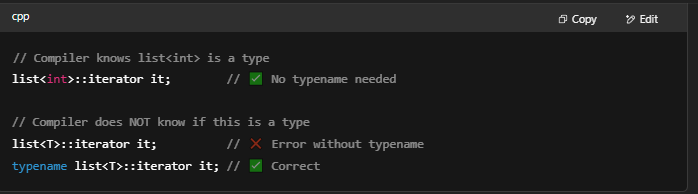
****

****

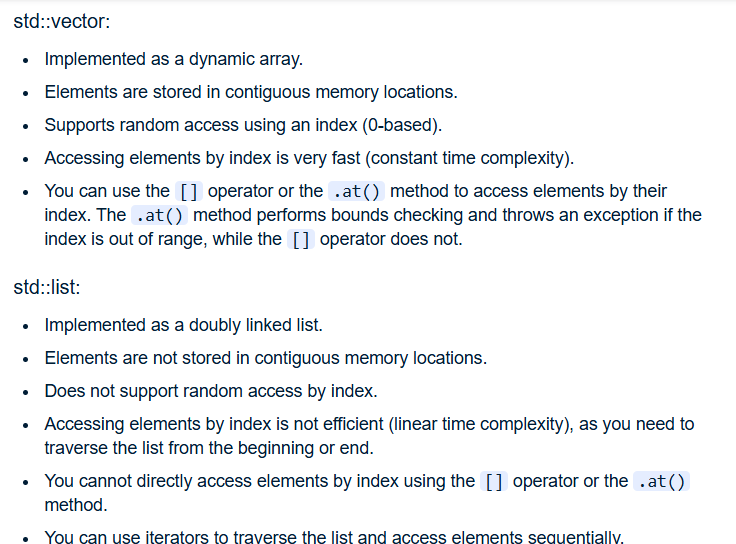
****

****

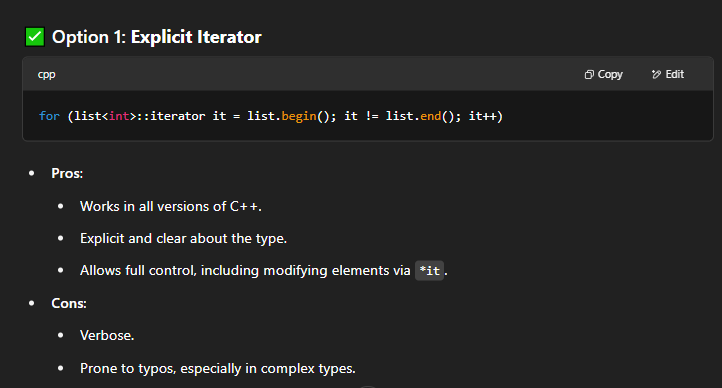
****

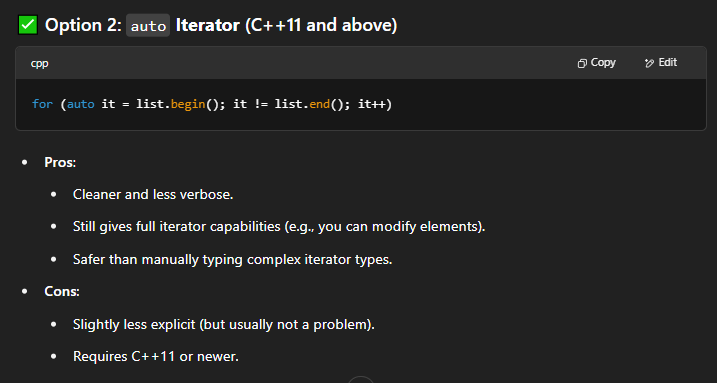
****

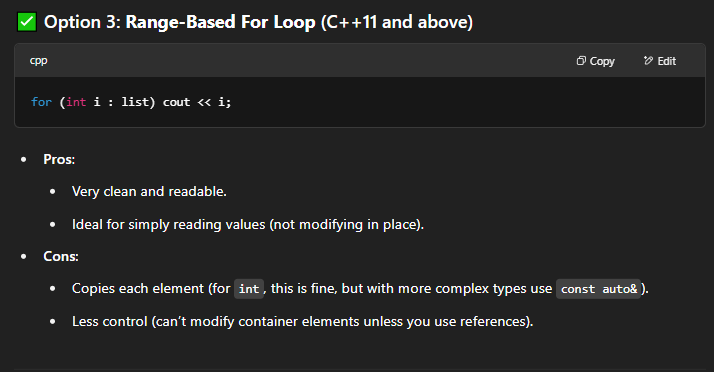
**EXTRA -->**

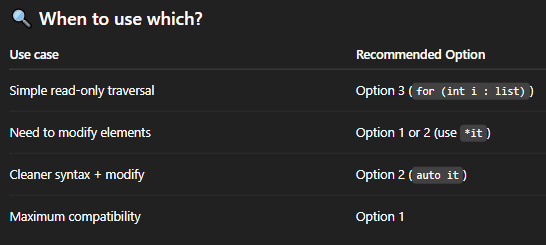
****

**OUTPUT METHODS 🡪**

****

****

****

****