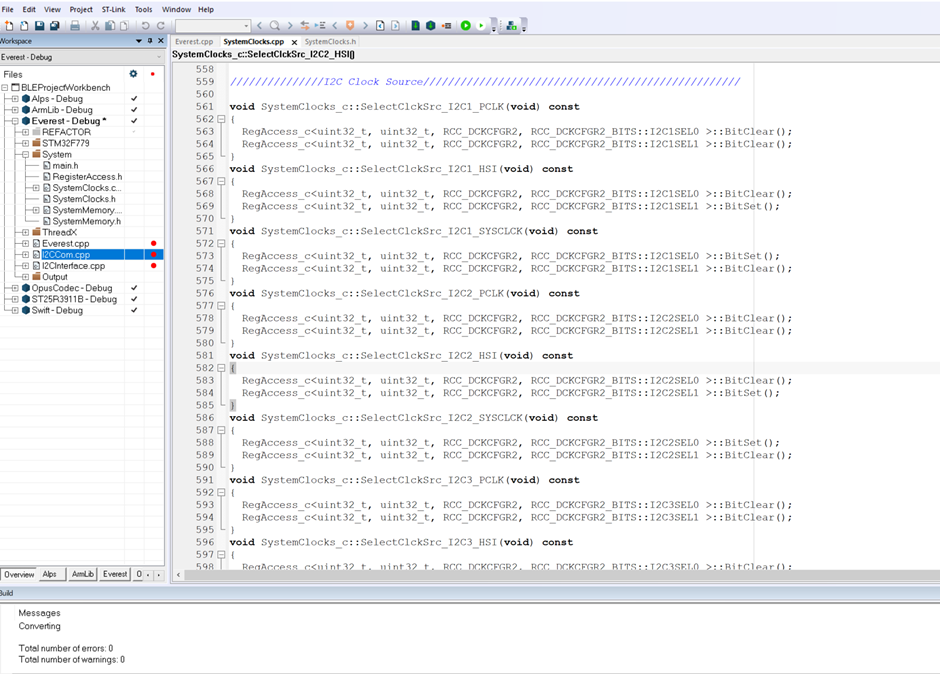
IAR Project from scratch:

***Everest Project from scratch***



// A C++ code to demonstrate that we can define

// methods outside namespace.

#include <iostream>

using namespace std;

// Creating a namespace

namespace ns

{

struct test{

int x;

int y;

};

void display();

class geek

{

public:

void display();

};

}

//using namespace ns;

// Defining methods of namespace

void ns::geek::display()

{

cout << "ns::geek::display()\n";

}

void ns::display()

{

cout << "ns::display()\n";

}

// Driver code

int main()

{

ns::test testing;

ns::geek obj;

ns::display();

obj.display();

testing.x= 5;

testing.y=6;

cout << "\n"<<testing.x;

return 0;

}

String parser:

#include <iostream>

#include <cstring>

//#include <stdio.h>

using namespace std;

int main()

{

// unsigned int j= 0;

cout << "Hello World" << endl;

char str[] ="- This, a sam/ple str@ing.Te-st";

char \* pch;

printf ("Splitting string \"%s\" into tokens:\n",str);

pch = strtok (str," ,.-1");

while (pch != NULL)

{

printf ("%s\n",pch);

pch = strtok (NULL, " ,@.-");

}

return 0;

}

**Function Templates**

// Example program

#include <iostream>

#include <string>

#include <array>

using namespace std;

template<typename T>

void fill\_array( T size,T \*my\_array)

{

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

{

my\_array[i]= (i+1);

std::cout << my\_array[i];

}

// return new\_array;

return ;

}

int main()

{

int result\_array[21] = {0};

std::cout << "!\n";

fill\_array<int>(sizeof(result\_array) / sizeof(int),result\_array);

std::cout << "!\n";

for(int i=0;i<20;i++)

{

std::cout << result\_array[i];

}

}

**Singleton Design pattern**

#include <iostream>

#include <string>

using namespace std;

class Singleton {

public:

static Singleton\* getSingleton() {

if (!instance) { // Only allow one instance of class to be generated.

instance = new Singleton;

}

return instance;

}

void sayHello(string str) {

cout << "Hello from singleton \n" << str;

}

private:

Singleton() {cout << "Hello from singleton constructor\n";} //Private so cannot be invoked.

Singleton& operator=(Singleton const& rhs) {} //Private so cannot be invoked.

Singleton( Singleton const& ) {};

static Singleton\* instance;

};

class SingleTimer {

public:

static SingleTimer\* GetInstance(){

if(!instance)

{

instance = new SingleTimer;

}

return instance;

}

void AddNumber(void)

{

number++;

}

int getNumber(void)

{

return number;

}

private:

SingleTimer() =default;

static SingleTimer\* instance;

SingleTimer(SingleTimer const&);

SingleTimer& operator=(SingleTimer const& );

int number ;

};

//Create 1 instance;

Singleton\* Singleton::instance = NULL;

SingleTimer\* SingleTimer::instance= NULL;

int main(int argc, char\* argv[]) {

//Get instance create single object.

Singleton::getSingleton()->sayHello("Hello world!!! \n");

SingleTimer::GetInstance()->AddNumber();

SingleTimer::GetInstance()->AddNumber();

cout << SingleTimer::GetInstance()->getNumber()<<"\n";

return 0;

}

# Array examples C++

#include <iostream>

using namespace std;

uint16\_t arr[]={0,1,2,3,4,5,6};

int main()

{

int fullArray = sizeof(arr);

int oneArrayElement= sizeof(arr[0]);

cout << "Size of fullArray is= " <<fullArray<< endl;

cout << "Size of oneArrayElement is= " <<oneArrayElement<< endl;

int size = sizeof(arr)/sizeof(arr[0]);

cout << "Size is= " <<size<< endl;

return 0;

}

**Sorting of a vector**

// Example program

#include <iostream>

#include <string>

#include <vector>

void printVector(std::vector<int> a){

for (size\_t i=0; i <a.size(); i++) {

std::cout<<a[i]<<" ";

}

}

void bubbleSort(std::vector<int>& a)

{

size\_t j ;

for ( j=0 ; j < a.size()-1; j++)

{

unsigned int temp=0;

for (size\_t i = 0; i < a.size()-1-j; i++)

{

if (a[i]>a[i+1] )

{

temp= a[i];

a[i]= a[i+1];

a[i+1]= temp;

}

}

}

}

int main()

{

std::vector<int> myUnsortedData = {10,1,5,8,9,2,3,6,4,7};

std::string name;

std::cout << "What is your name? ";

// getline (std::cin, name);

std::cout << "Hello, " << name << "!\n";

for(auto i = myUnsortedData.begin();i!= myUnsortedData.end();++i)

{

std::cout << \*i << " ";

}

std::cout <<std::endl;

bubbleSort( myUnsortedData);

std::cout <<std::endl;

printVector(myUnsortedData);

std::cout <<std::endl;

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

{

std::cout << myUnsortedData[i] << " ";

}

}

Number repletion counter algorithm in an array:

#include <iostream>

#include <string>

#include <array>

int main ()

{

std::array<unsigned int,8> data\_set = {1,5,5,6,7,9,10,5};

int repetion\_counter=0;

for (unsigned int i=0;i<data\_set.size();i++)

{

std::cout << "Number is = "<< data\_set[i]<<std::endl;

for (unsigned int j=i+1; j<data\_set.size();j++)

{

if(data\_set[i] ==data\_set[j])

{

repetion\_counter++;

}

}

}

std::cout << "Number of repetion of nums is = “;

std::cout << repetion\_counter << “! \n";

}

Rule of Three and Rule of Five:

Assignment example:

#include <iostream>

#include <string>

using namespace std;

class Character

{

public:

Character()=default;

//const Character& operator = (const Character&)=delete ;

Character(string nm,int l,int st,int iq):name{nm},life{l},strength{st},IQ{iq}{}

string name;

int life;

int strength;

int IQ;

void print()

{

cout << this->name<<endl;

cout << this->life<<endl;

cout << this->strength<<endl;

cout << this->IQ<<endl;

}

};

int main() {

cout << "Hello World!\n";

Character c1("Bob",1,3,6);

Character c2("Builder",3,5,8);

cout << "Before assignment operation!\n";

c1.print();

c2.print();

c1=c2;

cout << "after assignment operation!\n";

c1.print();

c2.print();

}

Output:

Hello World!

Before assignment operation!

Bob

1

3

6

Builder

3

5

8

after assignment operation!

Builder

3

5

8

Builder

3

5

8

Deep copy constructor:

#include <iostream>

#include <cstring>

using namespace std;

class Character

{

friend std::ostream& operator<<(std::ostream& stream,const Character& string);

private:

char\* m\_Buffer;

unsigned int m\_Size;

public:

//Character()=default; Character& operator = (const Character&)=delete ;

Character(const char\* string)

{

m\_Size= strlen(string);

m\_Buffer = new char[m\_Size+1];

memcpy(m\_Buffer,string,m\_Size);

m\_Buffer[m\_Size] = 0;

}

Character(const Character& other):m\_Size(other.m\_Size)

{

m\_Buffer= new char[m\_Size+1];

memcpy(m\_Buffer, other.m\_Buffer, m\_Size+1);

cout << "copy constructor called " <<endl;

}

void print()

{

}

~Character()

{

delete[] m\_Buffer;

}

char& operator[](unsigned int index)

{

return m\_Buffer[index];

}

};

std::ostream& operator<<(std::ostream& stream,const Character& string)

{

stream <<string.m\_Buffer;

return stream;

}

void PrintString(const Character& string)

{

cout << string <<endl;

}

int main() {

cout << "Hello World!\n";

Character string = "Cherno";

Character second = string;

// second= "Test";

second[2]= 't';

cout << "Before assignment operation!\n";

cout<<string<<endl;

cout << "after assignment operation!\n";

cout<<second<<endl;

PrintString(string);

PrintString(second);

//std::cin.get();

}