UNLV | Howard R. Hughes College of Engineering

DASL-100.2
C++ Programming and Linux

DASL
Drones and Autonomous Systems Lab @ UNLV

Week 4-1

1. IDE (Integrated Development Environment)
2. Vector
3. Thread

Instructor: Truc Tran

02-18-2023

UNLV Howard R. Hughes College of Engineering

DASL-100.2
C++ Programming and Linux

DASL
Drones and Autonomous Systems Lab @ UNLV

1. IDE (Integrated Development Environment)

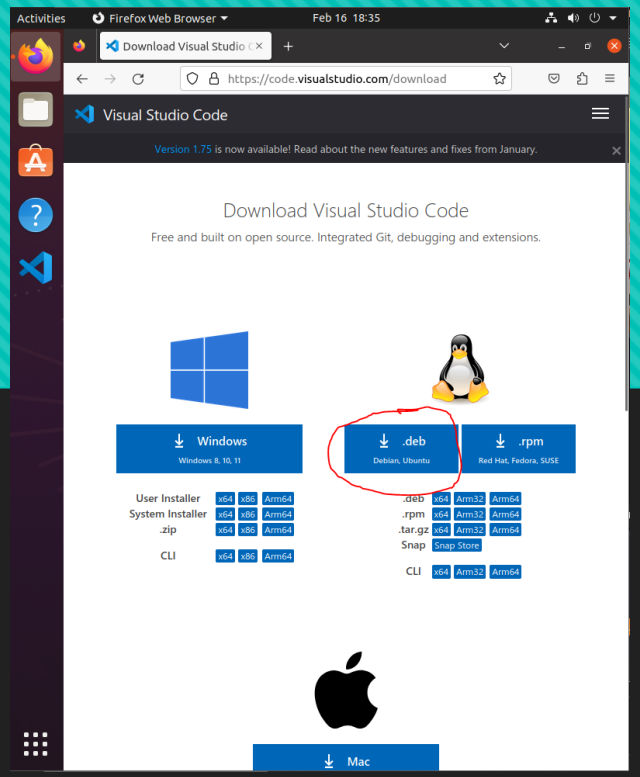
- IDE is a software application that integrates a code editor, compiler, debugger, and other tools to facilitate the development, testing, and debugging of C++ programs. It offers many features include:
  - Editing source code
  - Building executables, and debugging
  - Syntax highlighting, auto complete
- Common IDE are Visual Studio, Esclipse, Visual Studio Code, Code::Blocks or Clion.

Visual Studio | eclipse | Visual Studio Code | Code::Blocks The open source, cross-platform IDE | CLion A cross-platform IDE for C and C++

DASL-100.2
C++ Programming and Linux

1. IDE (Integrated Development Environment)

- We will be using Visual Studio Code:
  - Go to: https://code.visualstudio.com/download
  - Click on **.deb** button (This is for linux ubuntu)

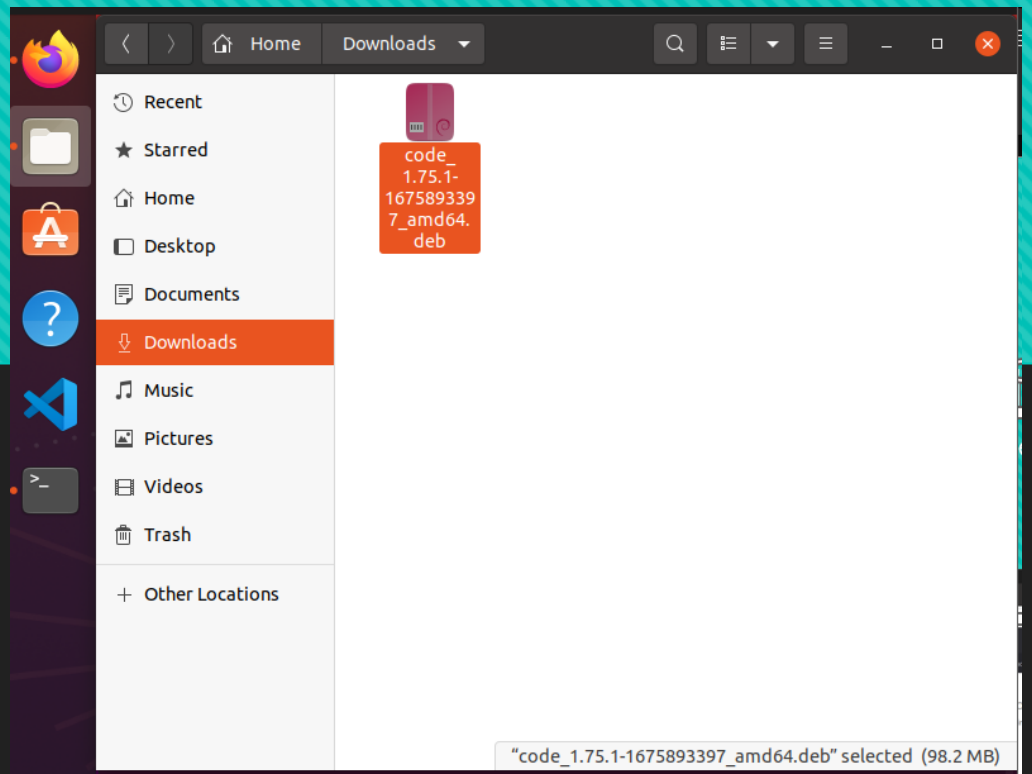


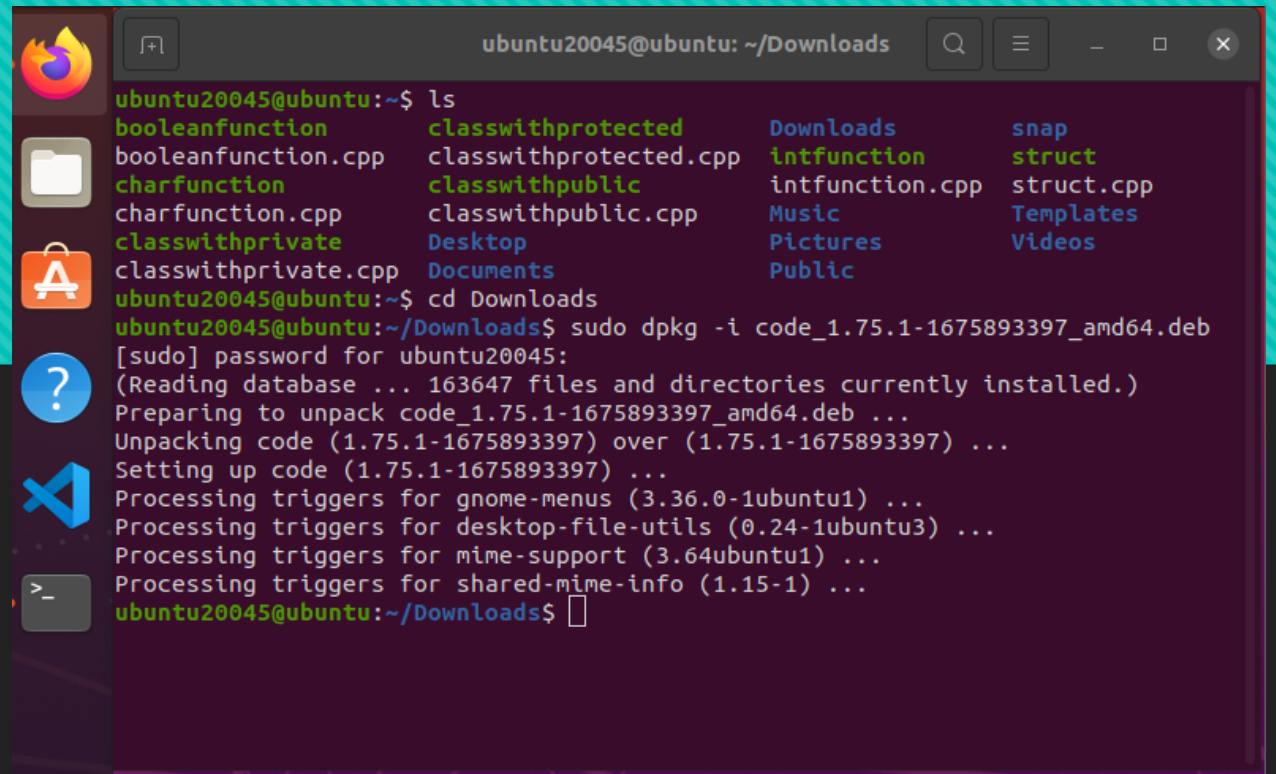
Visual Studio Code

DASL-100.2
C++ Programming and Linux

1. IDE (Integrated Development Environment)

- Installing Visual Studio Code in Ubuntu:
  - Check for the downloaded file in Downloads folder (code_1.75.1-1675893397_amd64.deb)
  - Open terminal and type in the command: sudo dpkg -i code_1.75.1-1675893397_amd64.deb

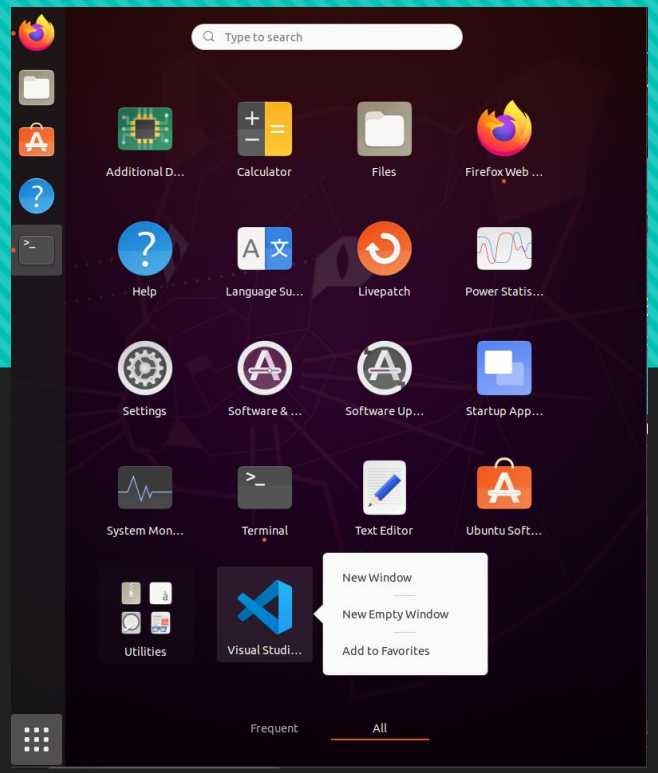


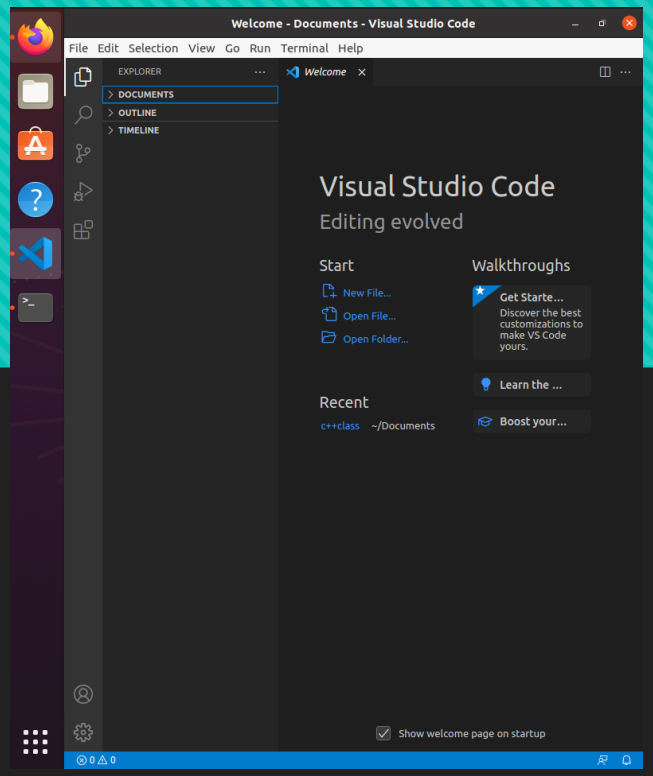


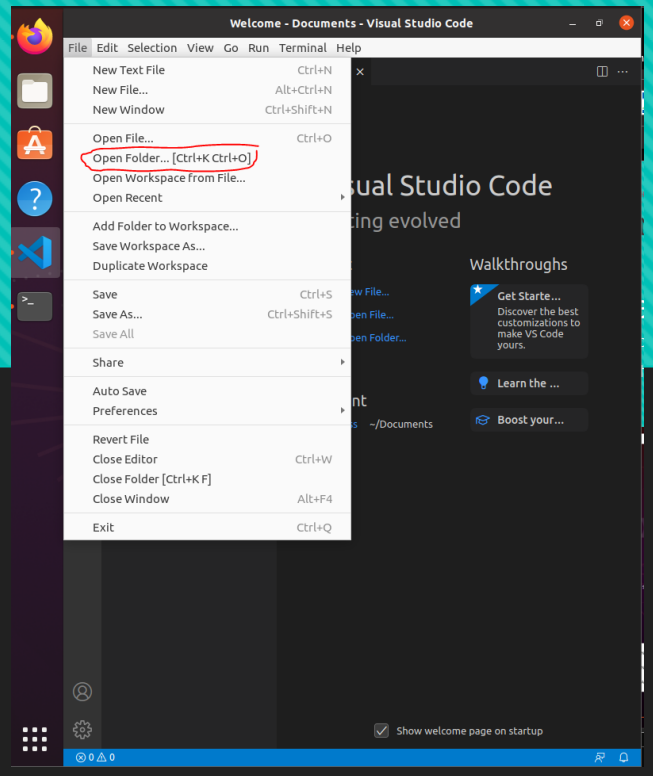
DASL-100.2
C++ Programming and Linux

1. IDE (Integrated Development Environment)

- Open applications and add Visual Studo Code to Favorite.
- Run Visual Studio Code > Click File > Click Open Folder



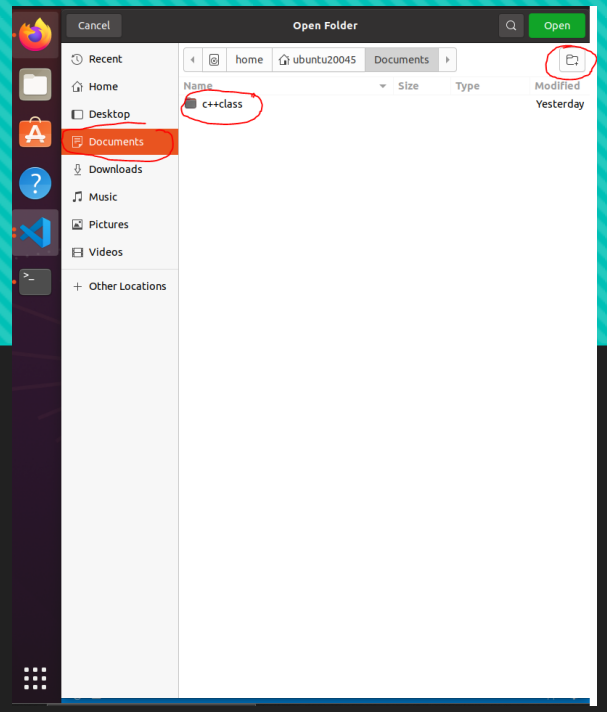


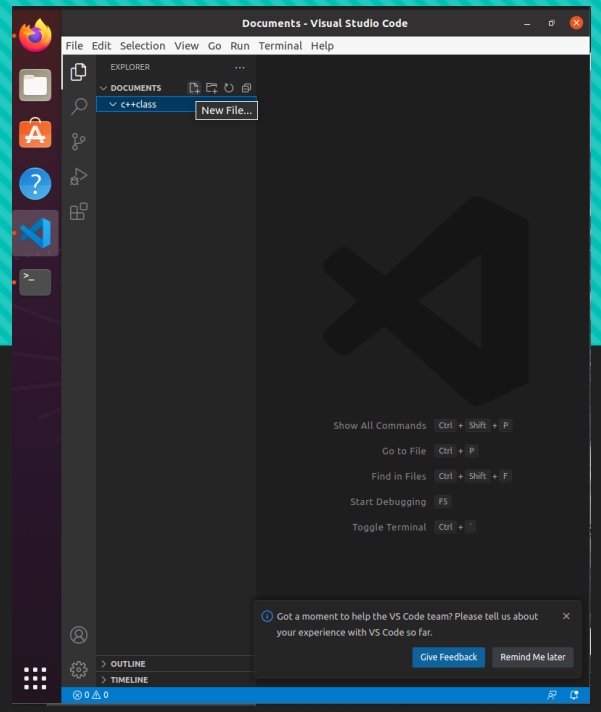


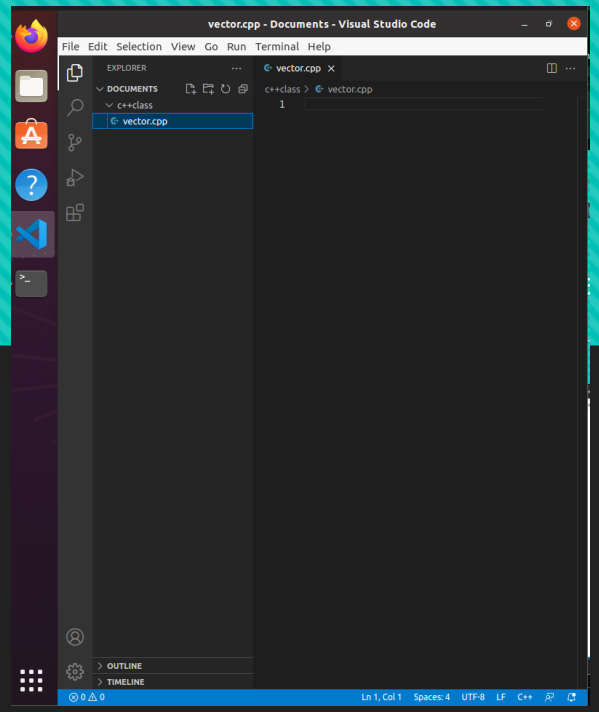
DASL-100.2
C++ Programming and Linux

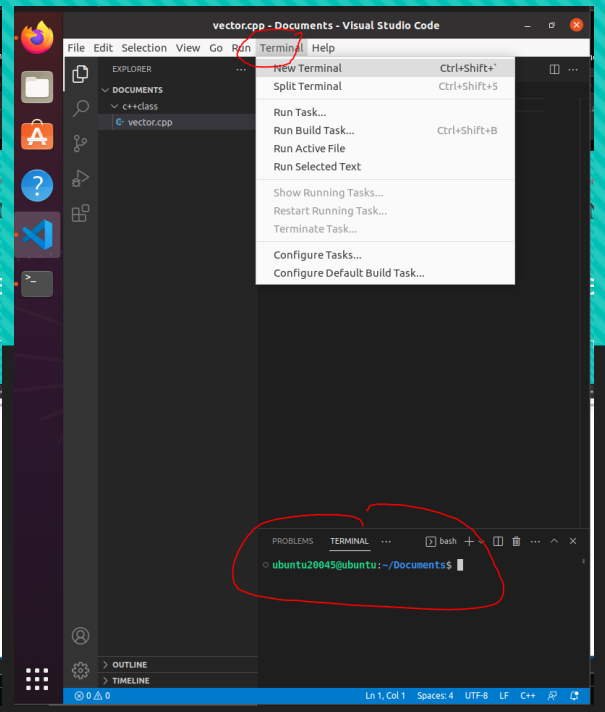
1. IDE (Integrated Development Environment)

- Navigate to Documents folder > Click add folder (top left icon) > Create a “c++ class” folder.
- Now we can create new .cpp files by clicking “New Files..”.
- The terminal can also be accessed in Visual Studio Code.









2. Vector

- Recall **array** is a collection of elements of the same data type stored in contiguous memory locations.
- In C++, **vector** is a dynamically resizable container that can store a collection of elements of a specific data type. Similar to **std::string**, **std::vector** is also a class that is part of the Standard Template Library (STL) and is defined in the <vector> header file. The common member functions of the **vector** class include:
  - push_back(): adds an element to the end of the vector.
  - pop_back(): removes the last element of the vector.
  - size(): returns the number of elements in the vector.
  - empty(): checks if the vector is empty or not. It returns true if the vector is empty and false otherwise.
  - clear(): removes all elements from the vector.
  - insert(): inserts an element at a specified position in the vector.
  - erase(): removes an element at a specified position in the vector.
  - front(): returns the first element of the vector.
  - back(): returns the last element of the vector.
  - begin(): returns an iterator to the beginning of the vector.
  - end(): return an integrator to the end of the vector.

UNLV Howard R. Hughes College of Engineering

DASL-100.2 C++ Programming and Linux

DASL Drones and Autonomous Systems Lab @ UNLV

2. Vector

Feature	Vector	Array
Size	Dynamic sizing	Size is fixed
Inserting elements	Efficient, with functions like push_back and insert	Not efficient, elements need to be shifted to make room for new ones
Removing elements	Efficient, with functions like pop_back and erase	Not efficient, elements need to be shifted to close the gap
Memory management	Managed automatically, memory is allocated and deallocated as needed	Managed manually, requires explicit allocation and deallocation of memory
Standard Library support	More built-in functions, more functionality	Fewer built-in functions, limited functionality

DASL-100.2
C++ Programming and Linux

2. Vector

vector.cpp - Documents - Visual Studio Co

File Edit Selection View Go Run Terminal Help

EXPLORER

DOCUMENTS
- c++class
  - vector
  - vector.cpp

vector.cpp

c++class > vector.cpp

```cpp
#include <iostream>
#include <vector>

int main(){
    std::vector<int> vec;

    std::cout << "Vector's size:" << vec.size() << std::endl;

    vec.push_back(10);
    vec.push_back(50);
    vec.push_back(100);

    std::cout << "Vector's size:" << vec.size() << std::endl;

    std::cout << "Vector's element 0 value: " << vec[0] << std::endl;
    std::cout << "Vector's element 1 value: " << vec[1] << std::endl;
    std::cout << "Vector's element 2 value: " << vec[2] << std::endl;

    std::cout << "Vector's first element value: " << vec.front() << std::endl;
    std::cout << "Vector's first last value: " << vec.back() << std::endl;

    vec.insert(vec.begin(), 0);

    std::cout << "Vector's size:" << vec.size() << std::endl;

    std::cout << "Vector's element 0 value: " << vec[0] << std::endl;
    std::cout << "Vector's element 1 value: " << vec[1] << std::endl;
    std::cout << "Vector's element 2 value: " << vec[2] << std::endl;
    std::cout << "Vector's element 3 value: " << vec[3] << std::endl;
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
ubuntu20045@ubuntu:~/Documents/c++class$
```

OUTLINE

TIMELINE

```cpp
    std::cout << "Vector's element 2 value: " << vec[2] << std::endl;

    std::cout << "Vector's first element value: " << vec.front() << std::endl;
    std::cout << "Vector's first last value: " << vec.back() << std::endl;

    vec.insert(vec.begin(), 0);

    std::cout << "Vector's size:" << vec.size() << std::endl;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
ubuntu20045@ubuntu:~/Documents/c++class$ g++ vector.cpp -o vector
ubuntu20045@ubuntu:~/Documents/c++class$ ls
vector  vector.cpp
ubuntu20045@ubuntu:~/Documents/c++class$ ./vector
Vector's size:0
Vector's size:3
Vector's element 0 value: 10
Vector's element 1 value: 50
Vector's element 2 value: 100
Vector's first element value: 10
Vector's first last value: 100
Vector's size:4
Vector's element 0 value: 0
Vector's element 1 value: 10
Vector's element 2 value: 50
Vector's element 3 value: 100
ubuntu20045@ubuntu:~/Documents/c++class$
```

OUTLINE

TIMELINE

⊗ 0 ⚠ 0

DASL-100.2
C++ Programming and Linux

3. Thread

- In C++, code is executed in a top-to-bottom flow, meaning that the statements in a program are executed in the order in which they appear. The code is compiled into machine code by the compiler and executed by the operating system, which reads the code line by line and performs the operations specified by each statement.
- In this example, "Read_Sensor_Data" function needs to wait for the "Read_Servo_Position" function to finish first before it can start. This can cause a delay in your robotic system. You wish your control system to operate in a continuous time frame and all the components are working "Independently".
- In C++, a **thread** is a lightweight execution context that can run in parallel with other threads. A thread can execute code independently from the main program, and multiple threads can run concurrently to perform tasks simultaneously.
- Threads are commonly used to improve the performance and responsiveness of programs by parallelizing workloads across multiple CPUs or processor cores.

```cpp
#include <iostream>
#include <unistd.h>

void Read_Servo_Position(){
    int value;
    for (int i = 0; i < 10; i++){
        usleep(100000);
        value = i;
        std::cout << "Servo position:" << value << std::endl;
    }
}

void Read_Sensor_Data(){
    int data;
    for (int i = 0; i < 10; i++){
       usleep(100000);
       data = i;
       std::cout << "Sensor data:" << data << std::endl;
    }
}

int main(){

    Read_Servo_Position();

    Read_Sensor_Data();

    return 0;
}
```

3. Thread

- The **std::thread** class is not specifically part of the STL, it is part of the C++ Standard Library, which includes a variety of libraries for different purposes. The common member functions of the **thread** class include:
  - thread(): the default constructor, which creates a thread object that is not associated with any thread of execution.
  - join(): a member function that blocks the calling thread until the associated thread has completed execution.
  - detach(): a member function that allows the associated thread to execute independently of the thread that created it.
  - get_id(): a member function that returns the unique identifier of the thread.
  - joinable(): a member function that returns true if the thread object is associated with a thread of execution, and false otherwise.

DASL-100.2
C++ Programming and Linux

3. Thread

- Let's demonstrate the code without using thread.

File Edit Selection View Go Run Terminal Help

EXPLORER

DOCUMENTS
- c++class
  - nothread.cpp
  - thread
  - thread.cpp
  - vector
  - vector.cpp

vector.cpp | thread.cpp | vector | nothread.cpp

c++class > nothread.cpp

```cpp
#include <iostream>
#include <unistd.h>

void Read_Servo_Position(){
    int value;
    for (int i = 0; i < 10; i++){
        usleep(100000);
        value = i;
        std::cout << "Servo position:" << value << std::endl;
    }
}

void Read_Sensor_Data(){
    int data;
    for (int i = 0; i < 10; i++){
       usleep(100000);
       data = i;
       std::cout << "Sensor data:" << data << std::endl;
    }
}

int main(){

    Read_Servo_Position();

    Read_Sensor_Data();

    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
ubuntu20045@ubuntu:~/Documents/c++class$
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
ubuntu20045@ubuntu:~/Documents/c++class$ g++ nothread.cpp -o nothread
ubuntu20045@ubuntu:~/Documents/c++class$ ls
nothread  nothread.cpp  thread  thread.cpp  vector  vector.cpp
ubuntu20045@ubuntu:~/Documents/c++class$ ./nothread
Servo position:0
Servo position:1
Servo position:2
Servo position:3
Servo position:4
Servo position:5
Servo position:6
Servo position:7
Servo position:8
Servo position:9
Sensor data:0
Sensor data:1
Sensor data:2
Sensor data:3
Sensor data:4
Sensor data:5
Sensor data:6
Sensor data:7
Sensor data:8
Sensor data:9
ubuntu20045@ubuntu:~/Documents/c++class$
```

OUTLINE

TIMELINE

DASL-100.2
C++ Programming and Linux

3. Thread

- Now using thread. Note that we need to link the “pthread” library using “-pthread” at the end of g++ complier.

```
File Edit Selection View Go Run Terminal Help
EXPLORER
DOCUMENTS
c++class
nothread.cpp
thread
thread.cpp
vector
vector.cpp
vector.cpp   thread.cpp   vector   nothread.cpp
c++class > thread.cpp
```

```cpp
#include <iostream>
#include <unistd.h>
#include <thread>

void Read_Servo_Position(){
    int value;
    for (int i = 0; i < 10; i++){
        usleep(100000);
        value = i;
        std::cout << "Servo position:" << value << std::endl;
    }
}

void Read_Sensor_Data(){
    int data;
    for (int i = 0; i < 10; i++){
       usleep(100000);
       data = i;
       std::cout << "Sensor data:" << data << std::endl;
    }
}

int main(){
    std::thread thread1(Read_Servo_Position);
    std::thread thread2(Read_Sensor_Data);

    thread1.join();
    thread2.join();

    return 0;
}
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL
ubuntu20045@ubuntu:~/Documents/c++class$ g++ thread.cpp -o thread -pthread
ubuntu20045@ubuntu:~/Documents/c++class$ ls
nothread.cpp  thread  thread.cpp  vector  vector.cpp
ubuntu20045@ubuntu:~/Documents/c++class$ ./thread
Servo position:0
Sensor data:0
Servo position:1
Sensor data:1
Sensor data:2
Servo position:2
Servo position:3
Sensor data:3
Sensor data:4
Servo position:4
Servo position:5
Sensor data:5
Sensor data:6
Servo position:6
Sensor data:7
Servo position:7
Sensor data:8
Servo position:8
Sensor data:9
Servo position:9
ubuntu20045@ubuntu:~/Documents/c++class$
```