MTRN3500
Computing Applications in Mechatronics Systems

Getting used to CLR Applications

T3 - 2020

Motivation

- We would like to use well developed libraries available under CLR (Common Language Runtime)
- We would like the memory management to be done by the system than by us.
- We would like to take advantage of speed of execution that comes with reduced STACK usage and increased HEAP usage.
- We can easily add events.
- We would like to have the reverse engineering made difficult.

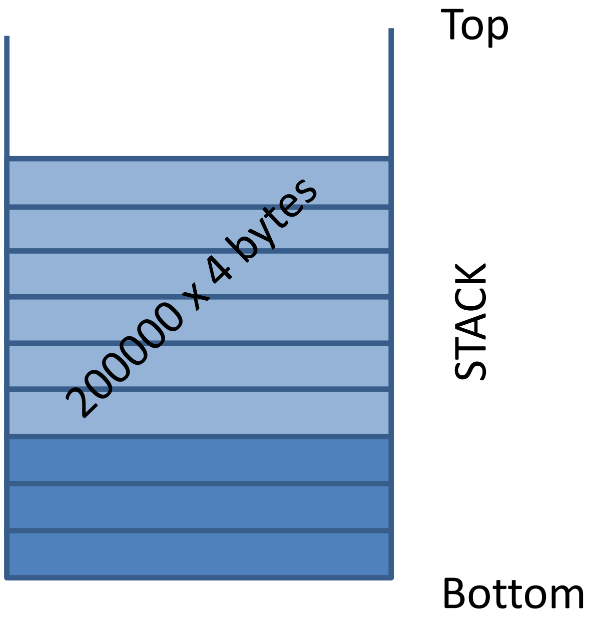
- LANGUAGE PROJECTIONS AVAILABLE
  - CLR
  - CX
  - WinRT

What are the differences?

- The C++ we have learnt so far is called NATIVE C++. Another word is UNMANAGED.
- We call it UNMANAGED because the system does not manage memory for us.
- You may recall, code fragments such as the one below.

```
int main()
{
    int Data[200000];  // Created on the STACK
    // Do some processing
    return 0;
}
```

- A bad habit is that we lavishly create objects requiring large chunks of memory, on the STACK. Anything between curly brackets is created on the stack.
- STACK has limited space and stack management is time consuming.
- Our preference is to minimize the stack usage.



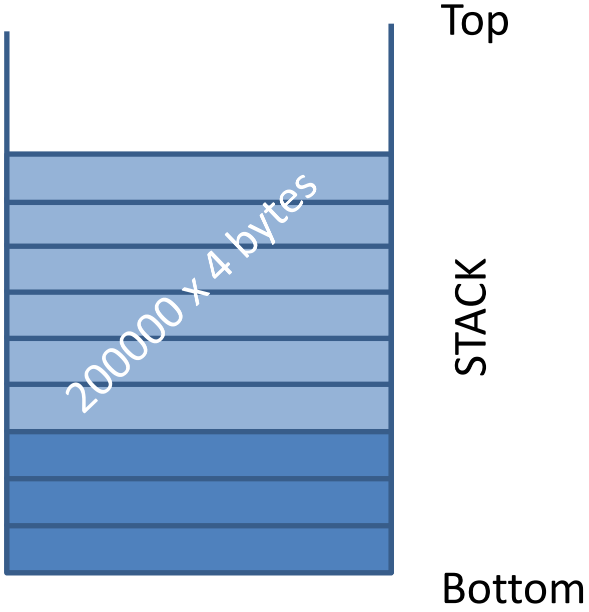
What are the differences?

- The C++ we have learnt so far is called NATIVE C++. Another word is UNMANAGED.
- We call it UNMANAGED because the system does not manage memory for us.
- You may recall, code fragments such as the one below.

```
int main()
{
    int Data[200000];  // Created on the STACK
    // Do some processing
    return 0;
}
```

← Close curly bracket

- A bad habit is that we lavishly create objects requiring large chunks of memory, on the STACK. Anything between curly brackets is created on the stack.
- STACK has limited space and stack management is time consuming.
- Our preference is to minimize the stack usage.



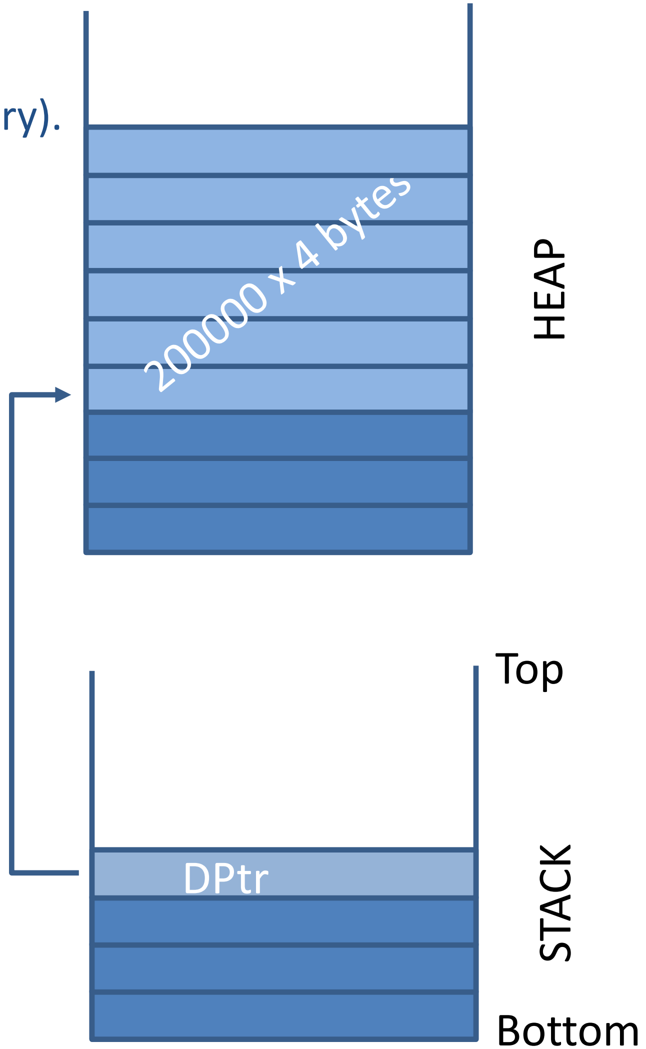
What are the differences?

- In C++ we have a way to not use STACK.
- We create data on the HEAP (on available UNUSED memory).
- We do it through dynamic memory allocation.

```
int main()
{
    //declare a pointer
    int *DPtr = NULL;
    //create data on heap
    DPtr = new int[200000];
    // Do some processing



    return 0;
}
```

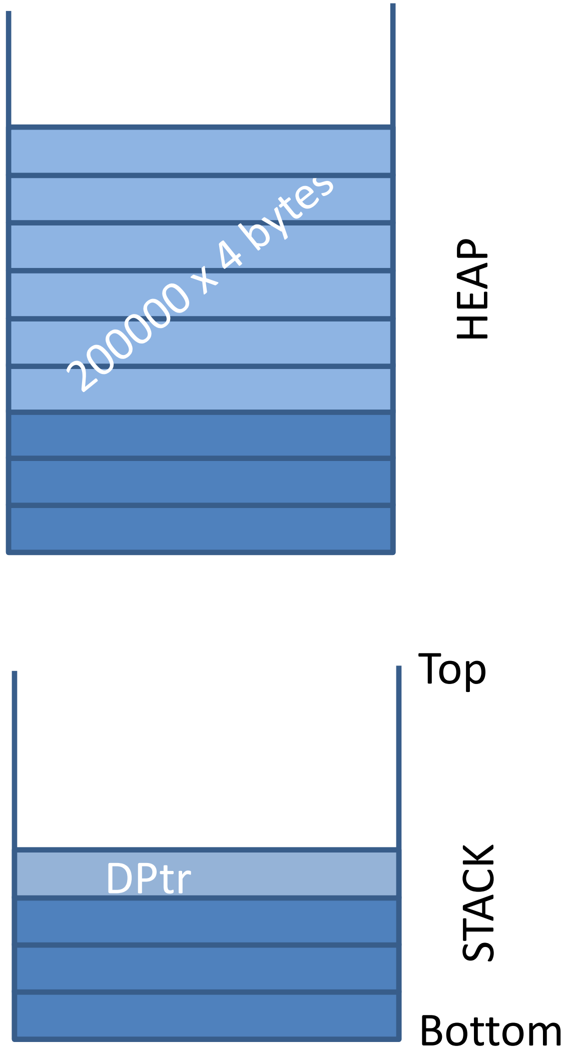


What are the differences?

- In C++ we have a way to not use STACK.
- We create data on the HEAP (on available UNUSED memory).
- We do it through dynamic memory allocation.

```
int main()
{
   //declare a pointer
   int *DPtr = NULL;
   //create data on heap
   DPtr = new int[200000];
   // Do some processing
  //free up space on heap
  if (Dptr != NULL)
         delete Dptr;
  return 0;
}
```

curly brace



What are the differences?

- The C++/CLR we want to use is called MANAGED.
- We call it MANAGED because the system does manage the memory for us.
- It also force us to use the HEAP by creating reference objects in contrast to value objects.

```
int main()
{
    int ^DPtr = nullptr;// This is how we declare HANDLES
    DPtr = gcnew <int>[200000];// we ask for memory on HEAP
    // It is not necessary to use delete
    return 0;
}
```

- It is preferred to create objects on the managed heap using gcnew.
- gc stands for garbage collector.
- Our preference is to minimize the stack usage.
- We can easily mix and match the native and managed code. Generally, mixed mode code is hard to reverse engineer.

A sample CLR program

```
#include <iostream> //Native
using namespace System;//Managed

int main()
{
       std::cout << "Program executed!" << std::endl;//Native
       Console::WriteLine("Program executed!");//Managed

       Console::ReadKey();//Managed
       return 0;//Native & Managed
}
```

A sample CLR program

```
#include <conio.h>

using namespace System;
using namespace System::Threading::Tasks;

int main()
{
     Console::WriteLine("Program started!");

     while (1)
     {
          System::Threading::Thread::Sleep(10);
          if (_kbhit()) break;
     }

     Console::ReadKey();
     Console::ReadKey();
     return 0;
}
```

A sample CLR program

```
#include <Windows.h>
#include <conio.h>

using namespace System;
using namespace System::Threading::Tasks;

int main()
{
	UINT64 Frequency, Counter = 0, OldCounter;
	double TimeStamp;
	Console::WriteLine("Program started!");
	QueryPerformanceFrequency((LARGE_INTEGER*)&Frequency);
	QueryPerformanceCounter((LARGE_INTEGER*)&OldCounter);

	while (1)
	{
		QueryPerformanceCounter((LARGE_INTEGER*)& Counter);
		TimeStamp = (double)(Counter - OldCounter) / Frequency;
		OldCounter = Counter;
		Console::WriteLine("{0, 12:F3} ", TimeStamp);
		System::Threading::Thread::Sleep(20);
		if (_kbhit()) break;
	}

	Console::ReadKey();
	Console::WriteLine(“Program Terminated Normally!”);
	Console::ReadKey();
	return 0;
}
```

A sample CLR program

```
#include <Windows.h>
#include <conio.h>

using namespace System;
using namespace System::Threading::Tasks;

int main(array<String^> ^args)
{
      UINT64 Frequency, Counter = 0, OldCounter;
      double TimeStamp;
      Console::WriteLine("Program started!");
      QueryPerformanceFrequency((LARGE_INTEGER*)&Frequency);
      QueryPerformanceCounter((LARGE_INTEGER*)&OldCounter);

      while (1)
      {
            QueryPerformanceCounter((LARGE_INTEGER*)& Counter);
            TimeStamp = (double)(Counter - OldCounter) / Frequency;
            OldCounter = Counter;
            Console::WriteLine("{0, 12:F3} ", TimeStamp);
            System::Threading::Thread::Sleep(20);
            if (_kbhit()) break;
      }

      Console::ReadKey();
      Console::ReadKey();
      return 0;
}
```

EVENTS

```
#using <System.dll>
#include <conio.h>

using namespace System;
using namespace System::Timers;

void OnElapsed(System::Object^ sender, System::Timers::ElapsedEventArgs^ e);

int main(array<String^>^ args)
{
        Timer^ MyTimer;
        Console::WriteLine("Program started!");
        MyTimer = gcnew Timer(500);
        //Timer configuration
        MyTimer->Enabled = true;
        MyTimer->AutoReset = true;
        MyTimer->Elapsed += gcnew System::Timers::ElapsedEventHandler(&OnElapsed);

        while (1)
        {
                if (_kbhit()) break;
        }
        MyTimer->Enabled = false;
        Console::ReadKey();
        Console::ReadKey();

        return 0;
}

void OnElapsed(System::Object^ sender, System::Timers::ElapsedEventArgs^ e)
{
        Console::WriteLine("Time now {0:HH:mm:ss.fff} ", e->SignalTime);
}
```