

Computer Programming

Dr. Deepak B Phatak
Dr. Supratik Chakraborty
Department of Computer Science and Engineering
IIT Bombay

Session: Pointers and Dynamic Memory – Part 1

Quick Recap of Relevant Topics



- Variables and pointers to variables in C++
- Storage in stack segment
- “Address of” operator: &
- “Content of” operator: *

Memory locations accessed: local variables/arrays of functions
Statically allocated in stack segment when function is called

Overview of This Lecture



- Storage in data segment (heap)
- Dynamic allocation of memory in C++
- Accessing dynamically allocated memory
- Good programming practices when using dynamic memory

Memory For Executing A Program (Process)

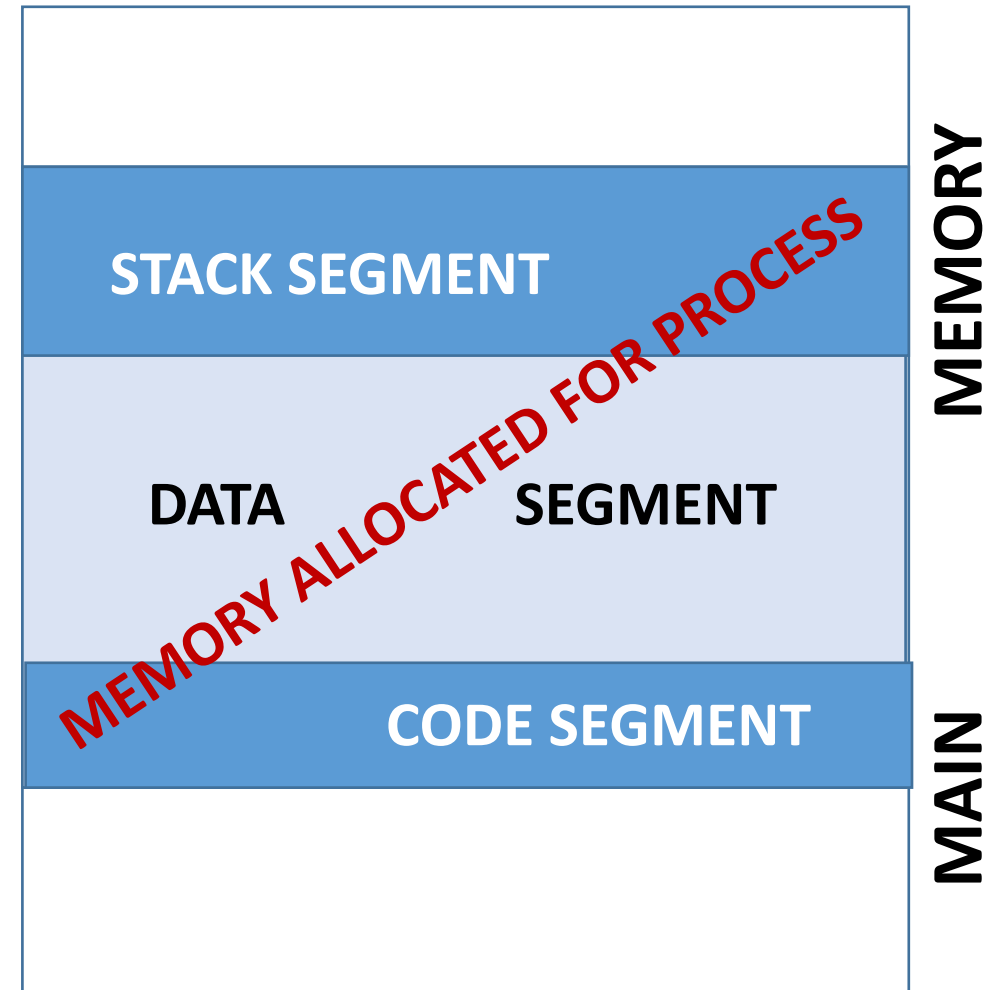
- Operating system allocates a part of main memory for use by a process

- Divided into:

Code segment: Stores executable instructions in program

Data segment: For dynamically allocated data (this lecture)

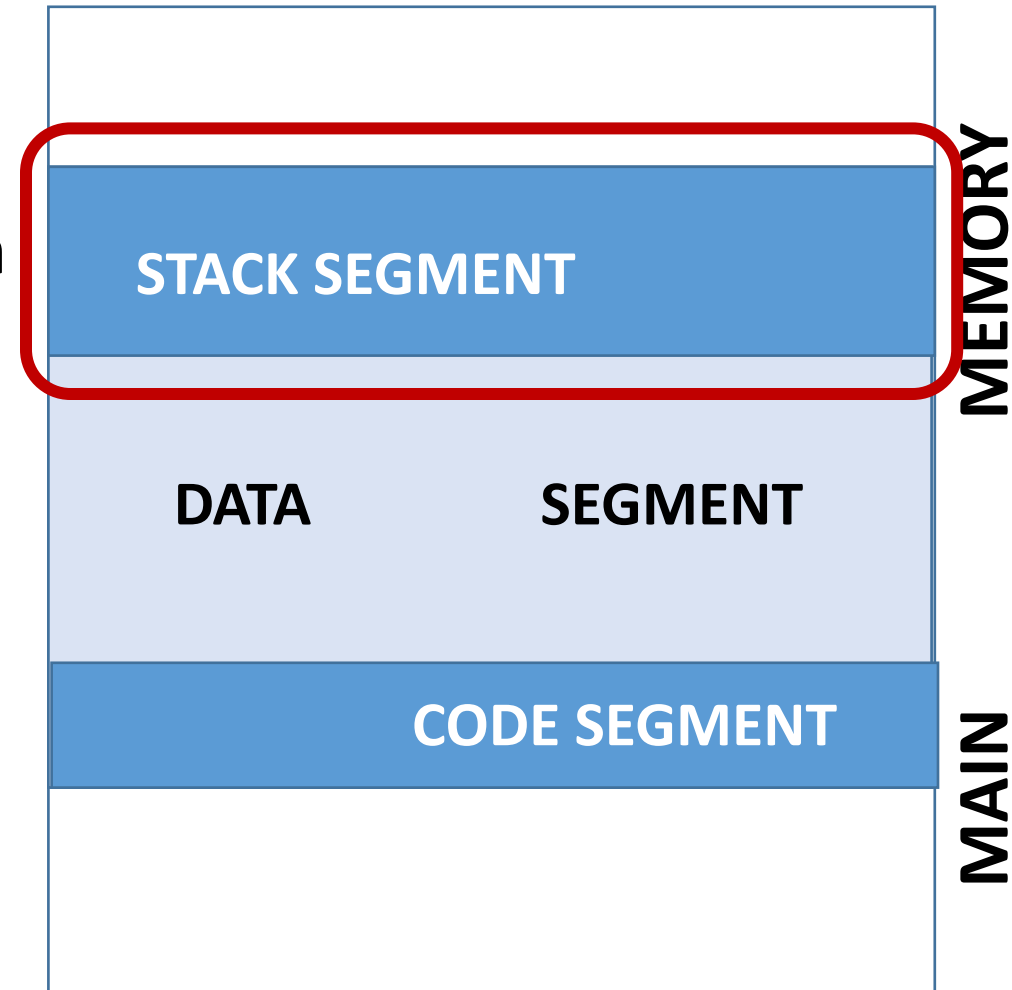
Stack segment: Call stack



Memory For Executing A Program (Process)

Local variables/arrays of a function

- Must be statically declared in function
- Memory allocated in activation record of function
- Resides in stack segment
- Ceases to exist once function ends and control returns to caller



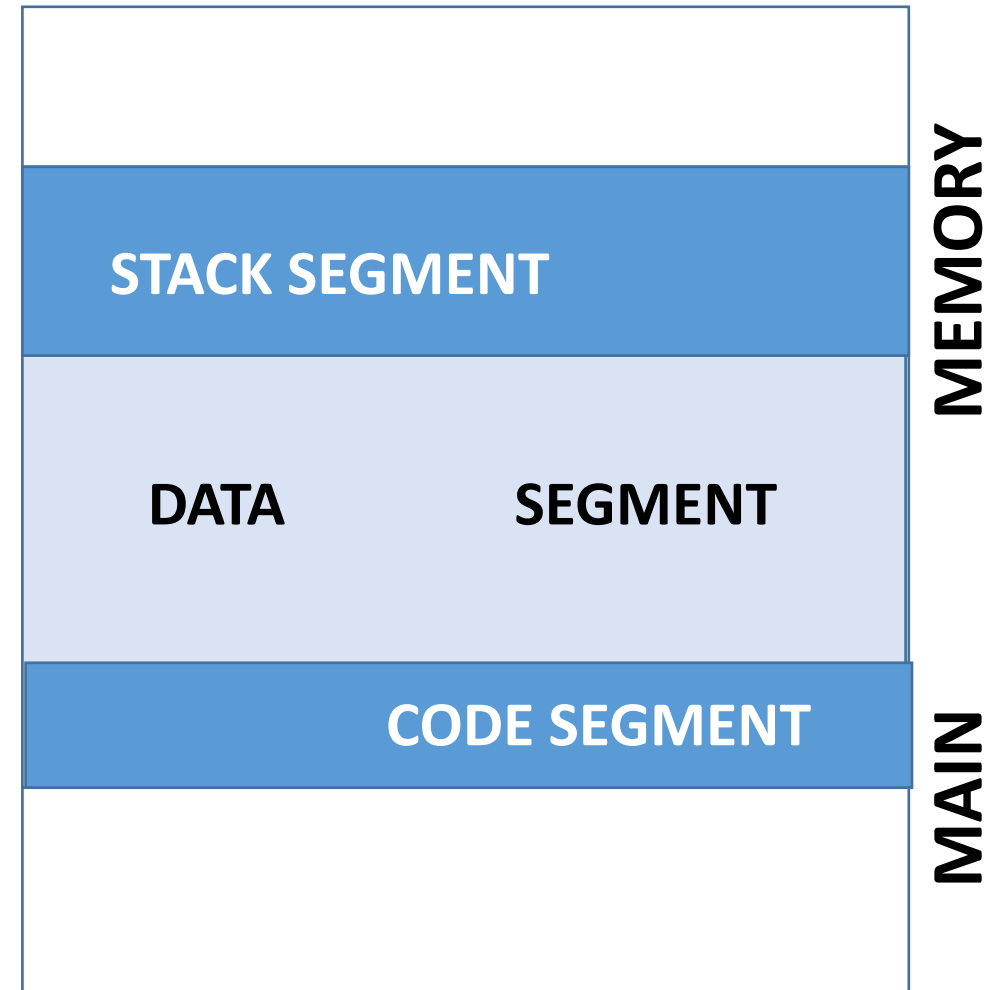
Memory For Executing A Program (Process)

What if the size of a local array is input-dependent?

Examples:

Read number of students in a class and store marks of all students in an int array.

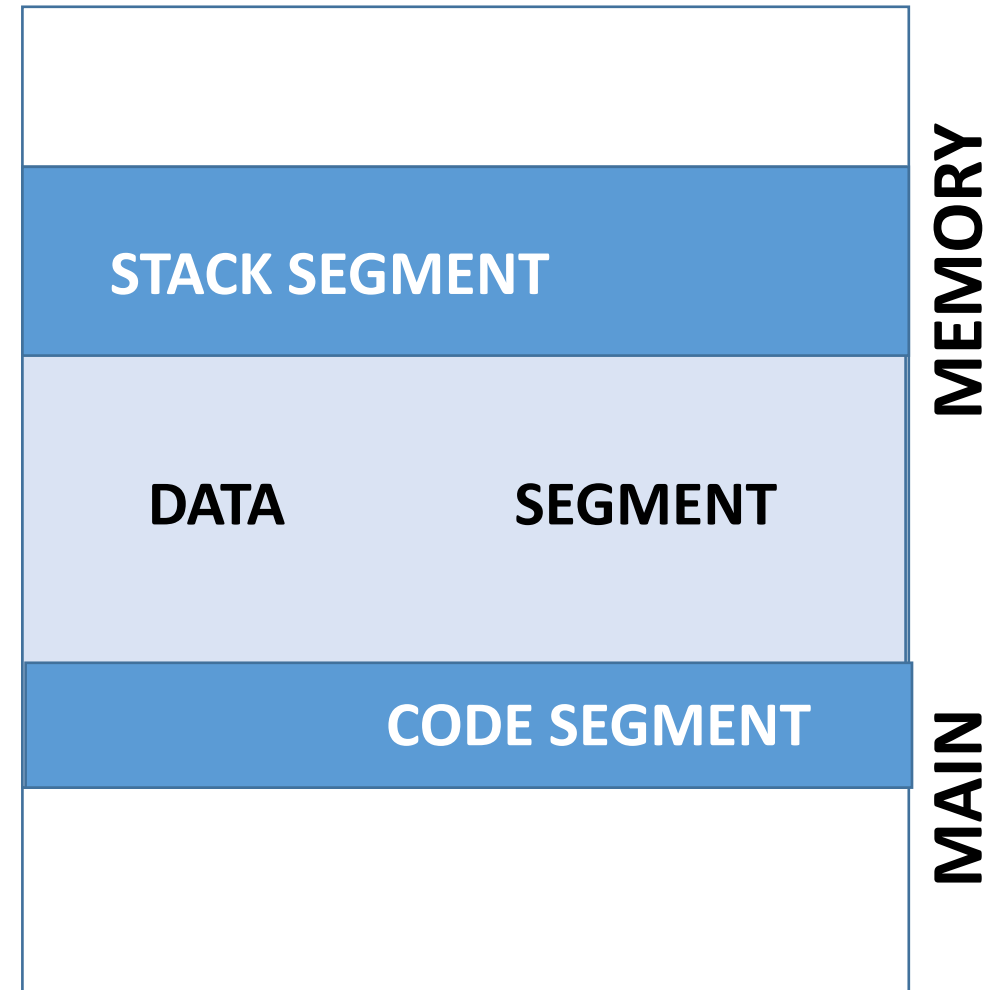
Read a sequence of characters ending with "." and store in a char array



Memory For Executing A Program (Process)

What if memory location(s) allocated in a function must be accessed after the function returns?

Examples: A function to read and store a sequence of characters in a char array, which must be used by the calling function



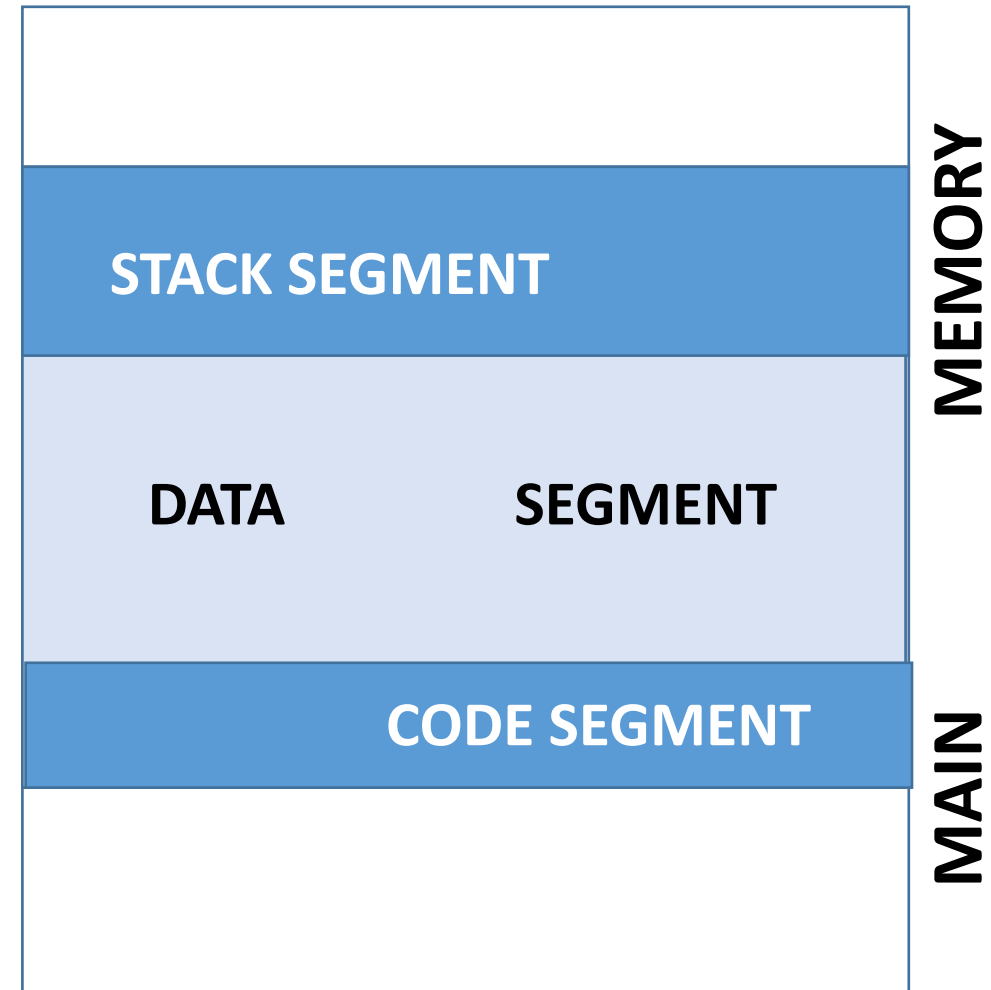
Memory For Executing A Program (Process)

We need a mechanism for allocating memory locations **dynamically**

Dynamic memory allocation:
Allocation at run time

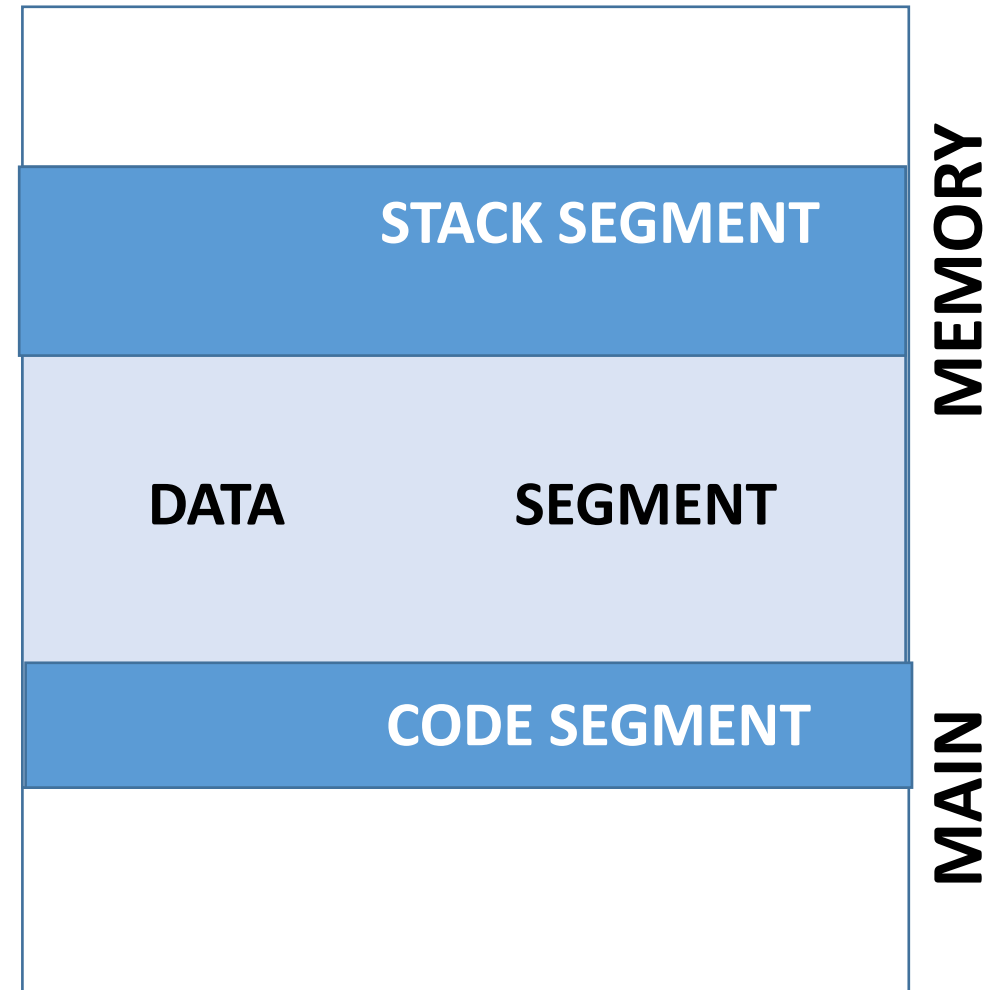
Allocation could depend on values of expressions read/computed

**[number of students,
number of chars in name...]**



Memory For Executing A Program (Process)

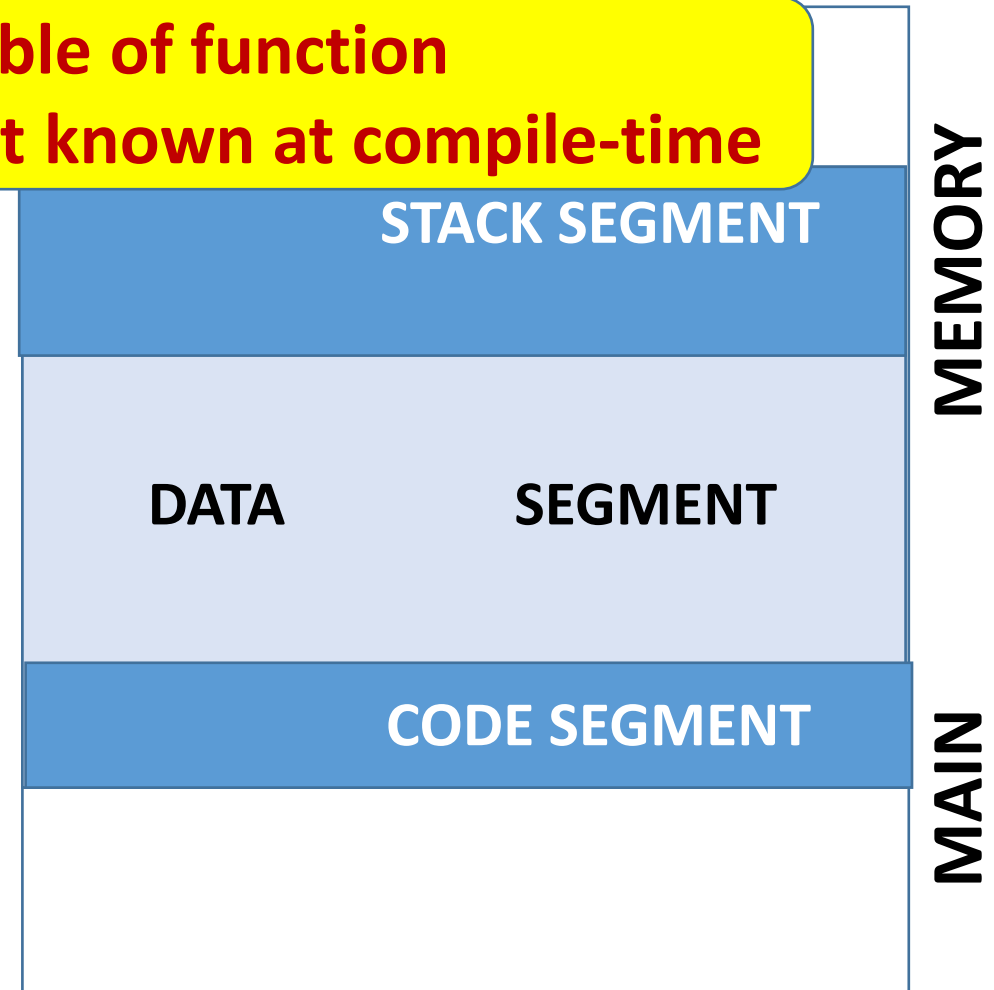
```
int main()
{
    int numStudents;
    cin >> numStudents;
    // Allocate an int array A
    // of size numStudents
    // Store quiz marks in A
    return 0;
}
```



Memory For Executing A Program (Process)

```
int main()
{
    int numStudents;
    cin >> numStudents;
    // Allocate an int array A
    // of size numStudents
    // Store quiz marks in A
    return 0;
}
```

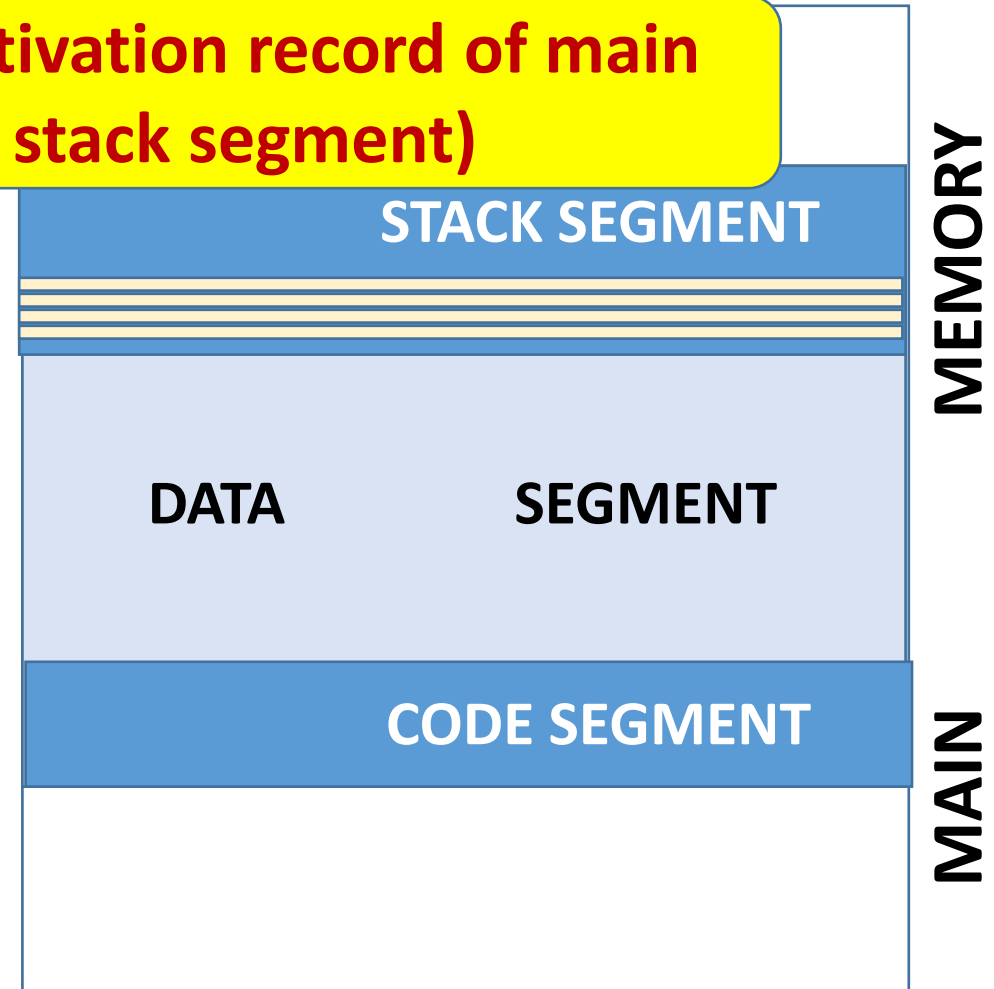
Local variable of function
Memory requirement known at compile-time



Memory For Executing A Program (Process)

```
int main()  
{  
    int numStudents;  
    cin >> numStudents;  
    // Allocate an int array A  
    // of size numStudents  
    // Store quiz marks in A  
    return 0;  
}
```

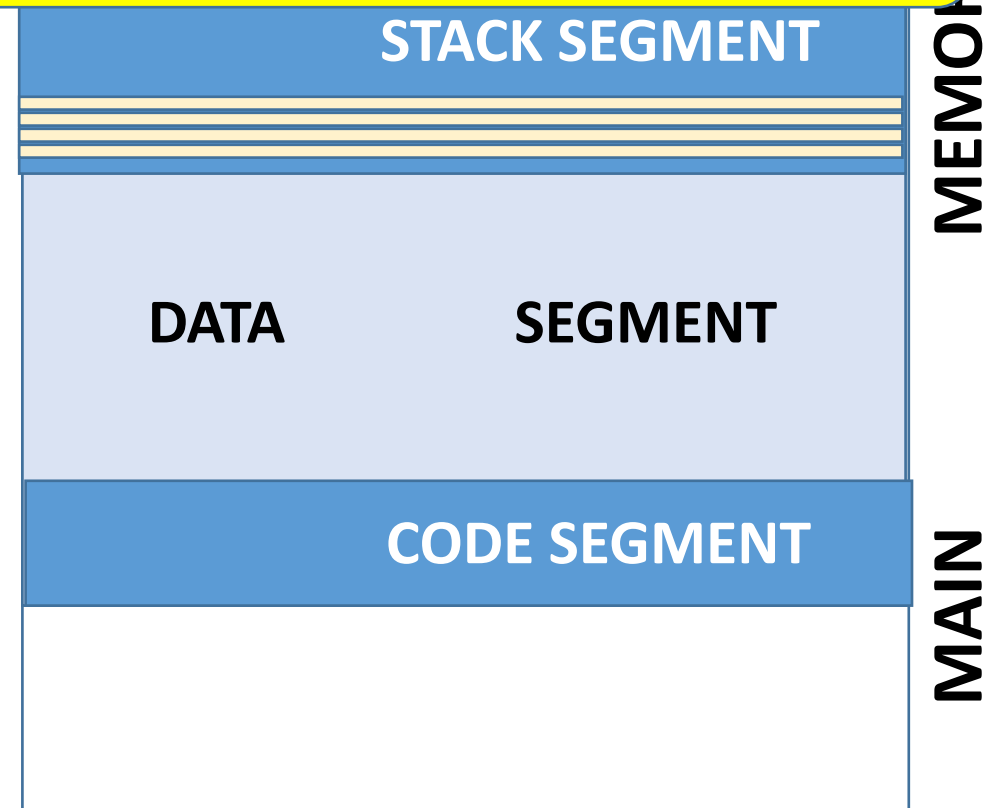
**Allocate space in activation record of main
(Call stack in stack segment)**



Memory For Executing A Program (Process)

```
int main()  
{  
    int numStudents;  
    cin >> numStudents;  
    // Allocate an int array A  
    // of size numStudents  
    // Store quiz marks in A  
    return 0;  
}
```

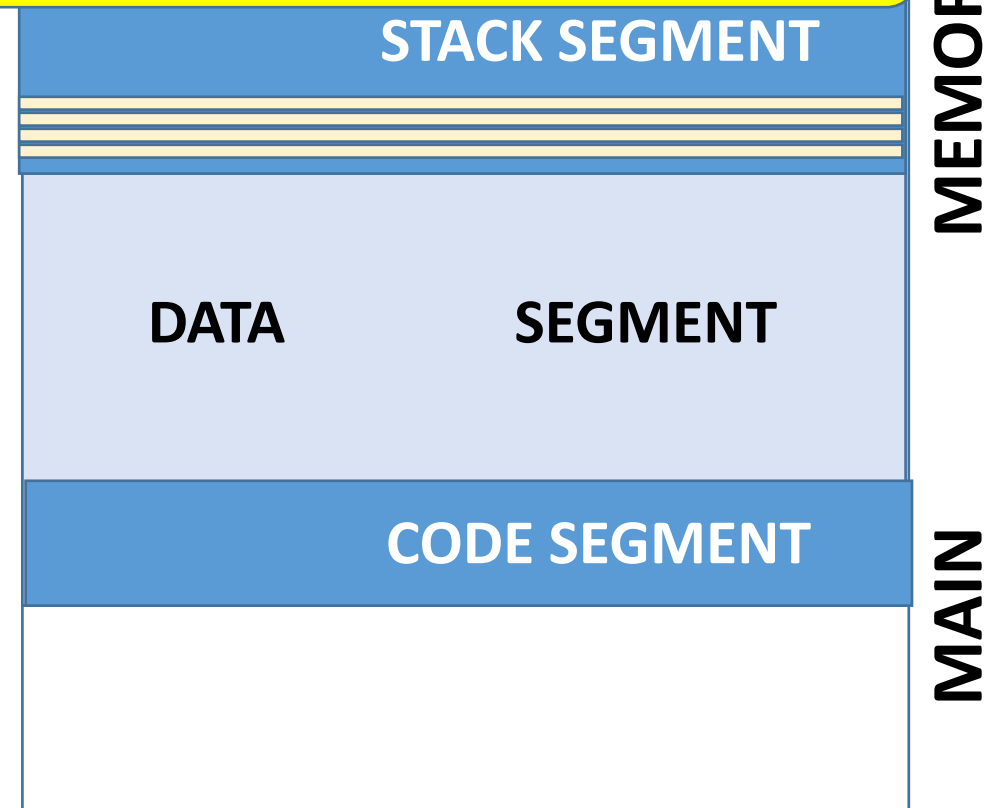
**Size of array A dependent on value of numStudents.
Memory requirement not known at compile time.**



Memory For Executing A Program (Process)

```
int main()  
{  
    int numStudents;  
    cin >> numStudents;  
    // Allocate an int array A  
    // of size numStudents  
    // Store quiz marks in A  
    return 0;  
}
```

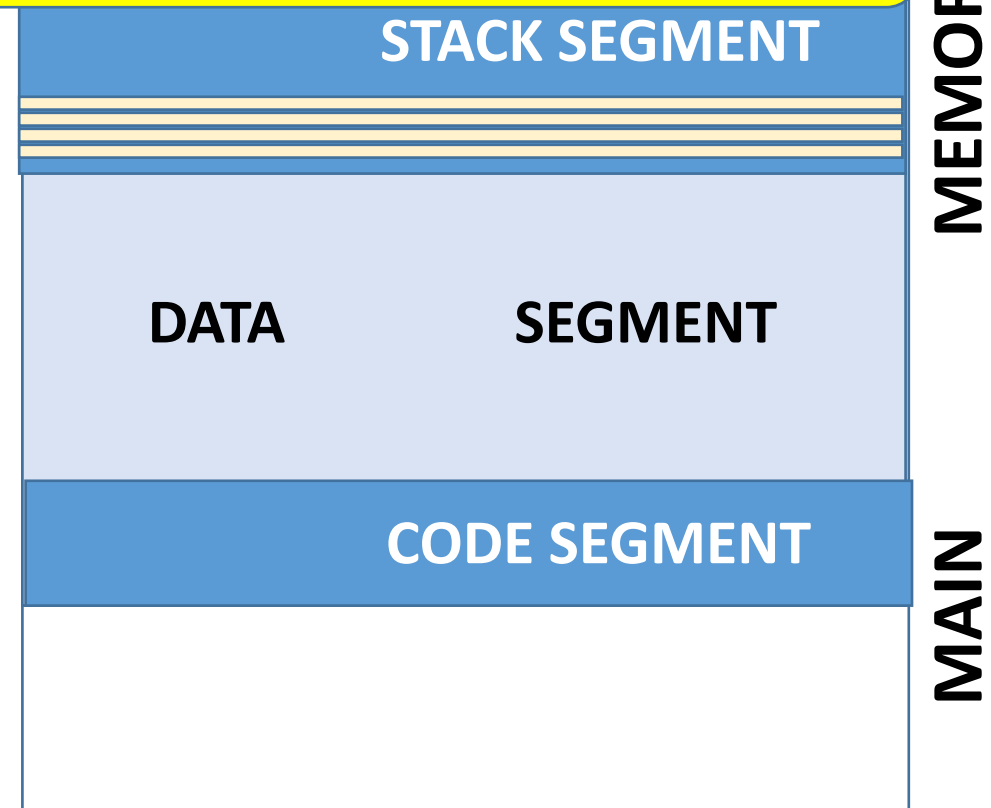
Cannot reserve space for array A when activation record of "main" is created in stack segment



Memory For Executing A Program (Process)

```
int main  
{  
    int numStudents;  
    cin >> numStudents;  
    // Allocate an int array A  
    // of size numStudents  
    // Store quiz marks in A  
    return 0;  
}
```

Where should this memory space come from?

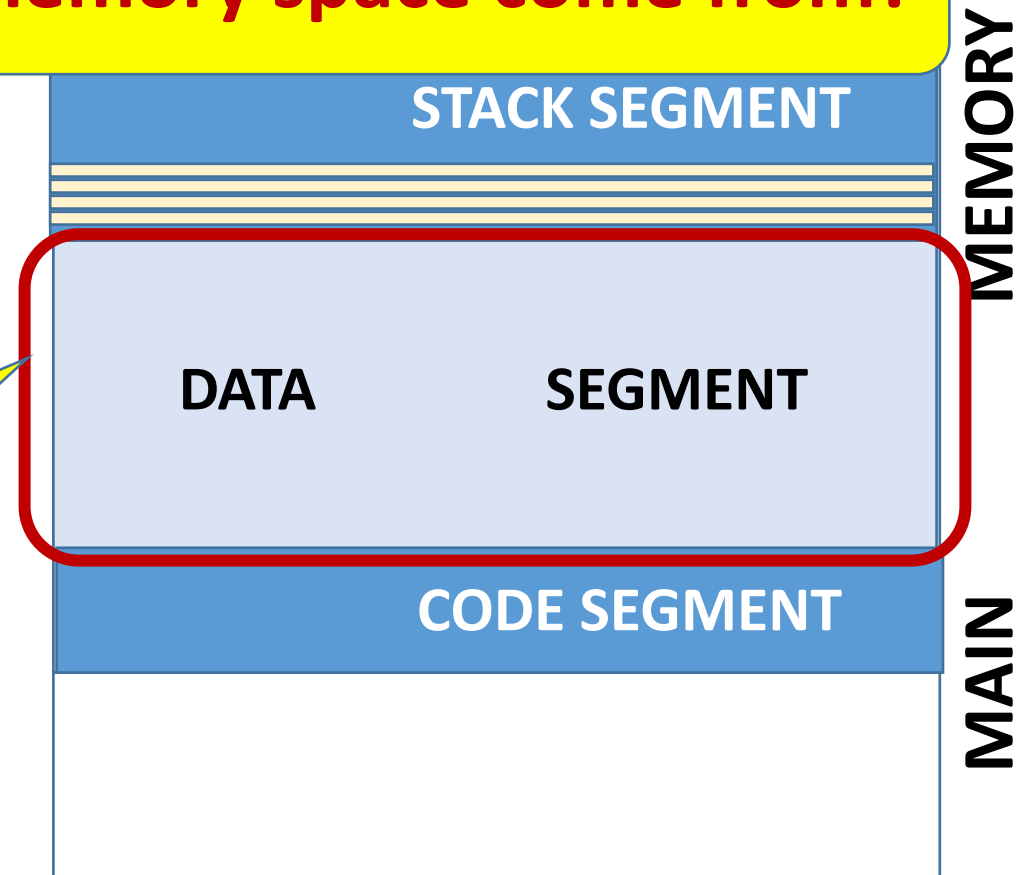


Memory For Executing A Program (Process)

```
int main  
{  
    int numStudent;  
    cin >> numStudents;  
    // Allocate an int array A  
    // of size numStudent  
    // S  
    return  
}
```

Where should this memory space come from?

**Data Segment
(also called “heap”)
to our rescue !**



Memory For Executing A Program (Process)

```
int main
```

```
{
```

```
    int numStud
```

```
    cin >> numStudents;
```

```
    // Allocate an int array A
```

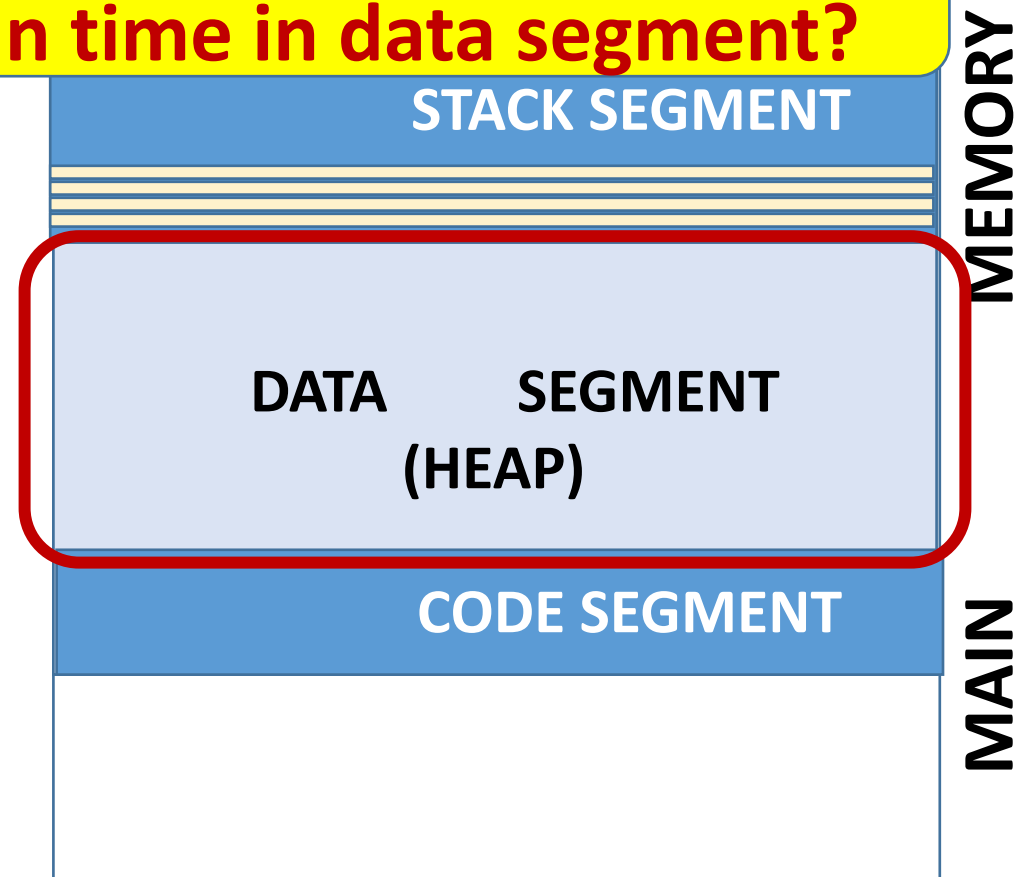
```
    // of size numStudents
```

```
    // Store quiz marks in A
```

```
    return 0;
```

```
}
```

How do we allocate an array of size numStudents at run time in data segment?



Dynamic Memory Allocation in C++

- C++ provides a special construct for dynamically allocating memory in heap (data segment)

```
A = new int [numStudents];
```

Dynamically allocate memory on heap

Dynamic Memory Allocation in C++

- C++ provides a special construct for dynamically allocating memory in heap (data segment)

Each element is an int

Allocate space for an array of size "numStudents"

```
A = new int [numStudents];
```

Dynamically allocate memory on heap

Dynamic Memory Allocation in C++

- C++ provides a special construct for dynamically allocating memory in heap (data segment)

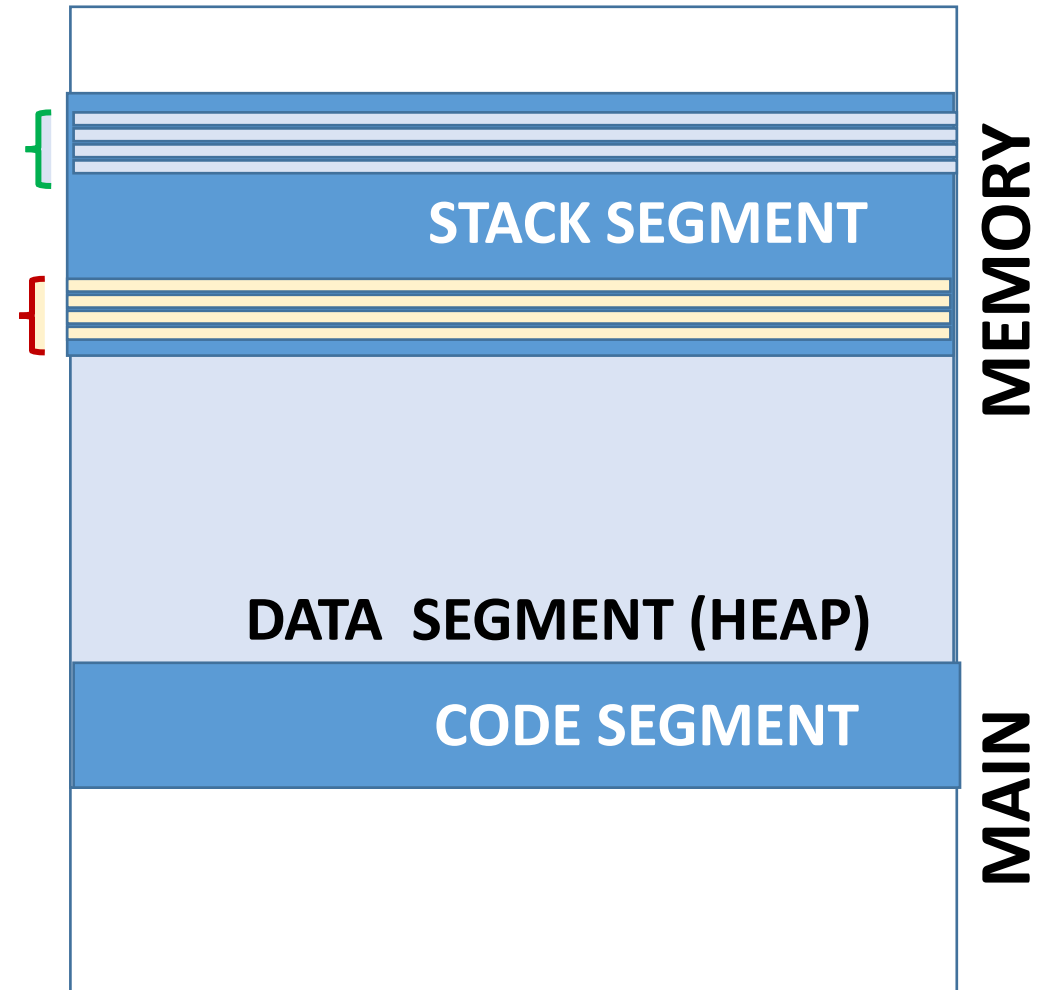
Returns pointer to an int

```
A = new int [numStudents];
```

A should be of type int *

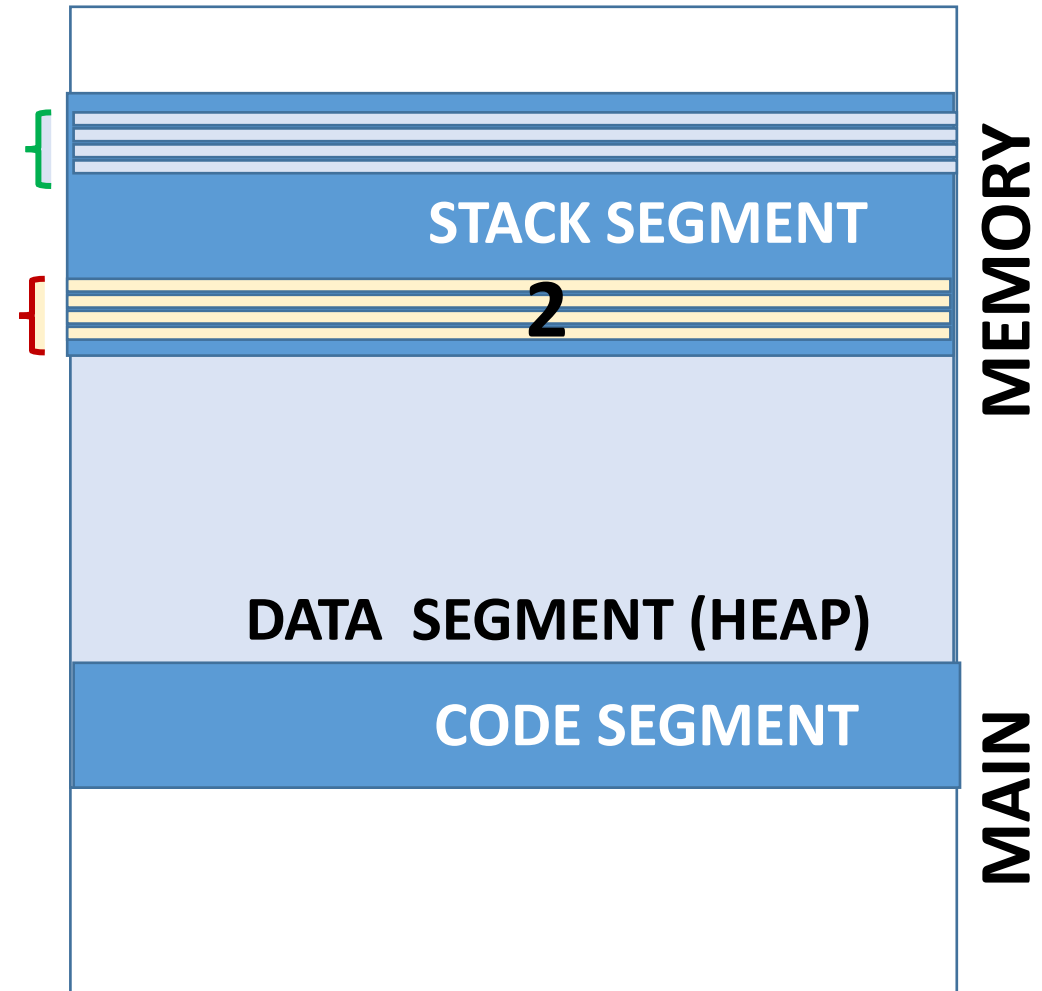
Memory For Executing A Program (Process)

```
int main()
{
    int numStudents;
    int * A;
    cin >> numStudents;
    A = new int[numStudents];
    // Store quiz marks in A
    return 0;
}
```



Memory For Executing A Program (Process)

```
int main()
{
    int numStudents;
    int * A;
    cin >> numStudents;
    A = new int[numStudents];
    // Store quiz marks in A
    return 0;
}
```



Memory For Executing A Program (Process)

```
int main()
```

```
{
```

```
    int numStudents;
```

```
    int * A;
```

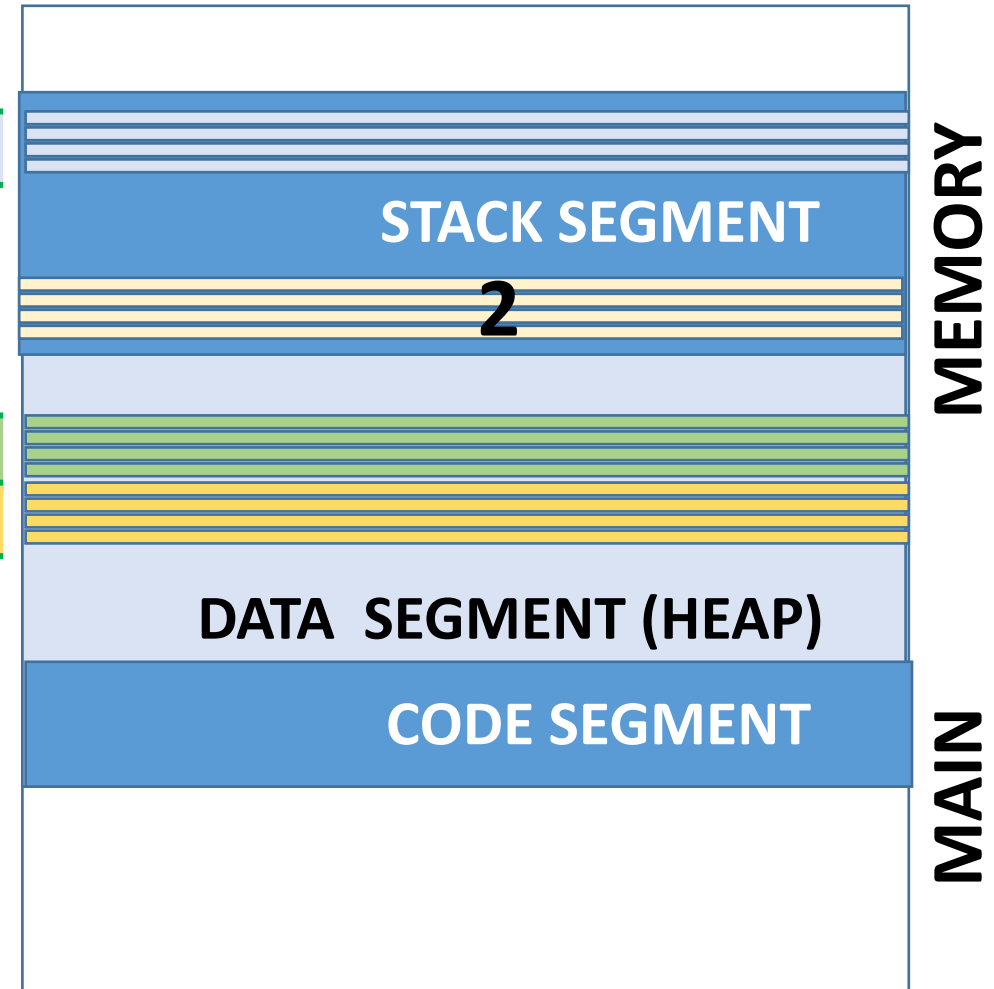
```
    cin >> numStudents;
```

```
    A = new int[numStudents];
```

```
    // Store quiz marks in A
```

```
    return 0;
```

```
}
```



Memory For Executing A Program (Process)

```
int main()
```

```
{
```

```
    int numStudents;
```

```
    int * A;
```

```
    cin >> numStudents;
```

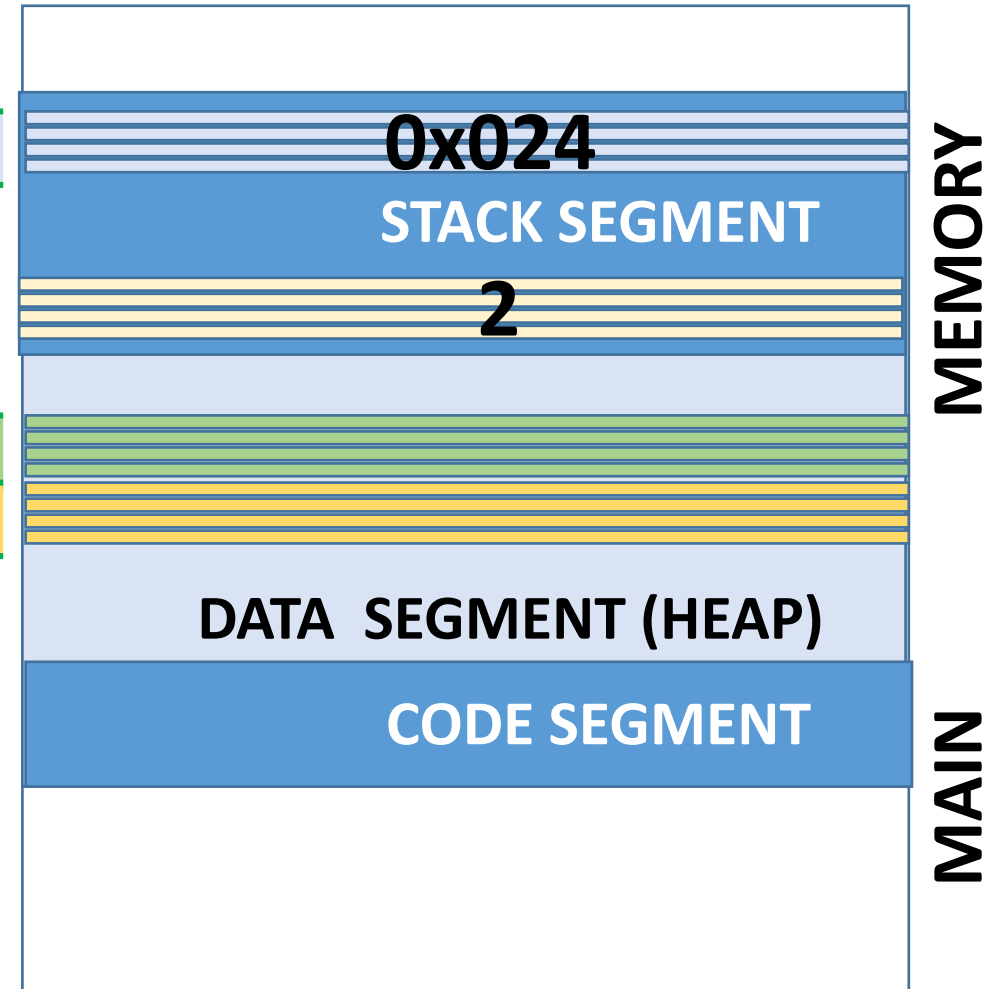
```
    A = new int[numStudents];
```

```
    // Store quiz marks in A
```

```
    return 0;
```

```
}
```

Address
0x024



Memory For Executing A Program (Process)

```
int main()
```

```
{
```

```
    int numStudents;
```

```
    int * A;
```

```
    cin >> numStudents;
```

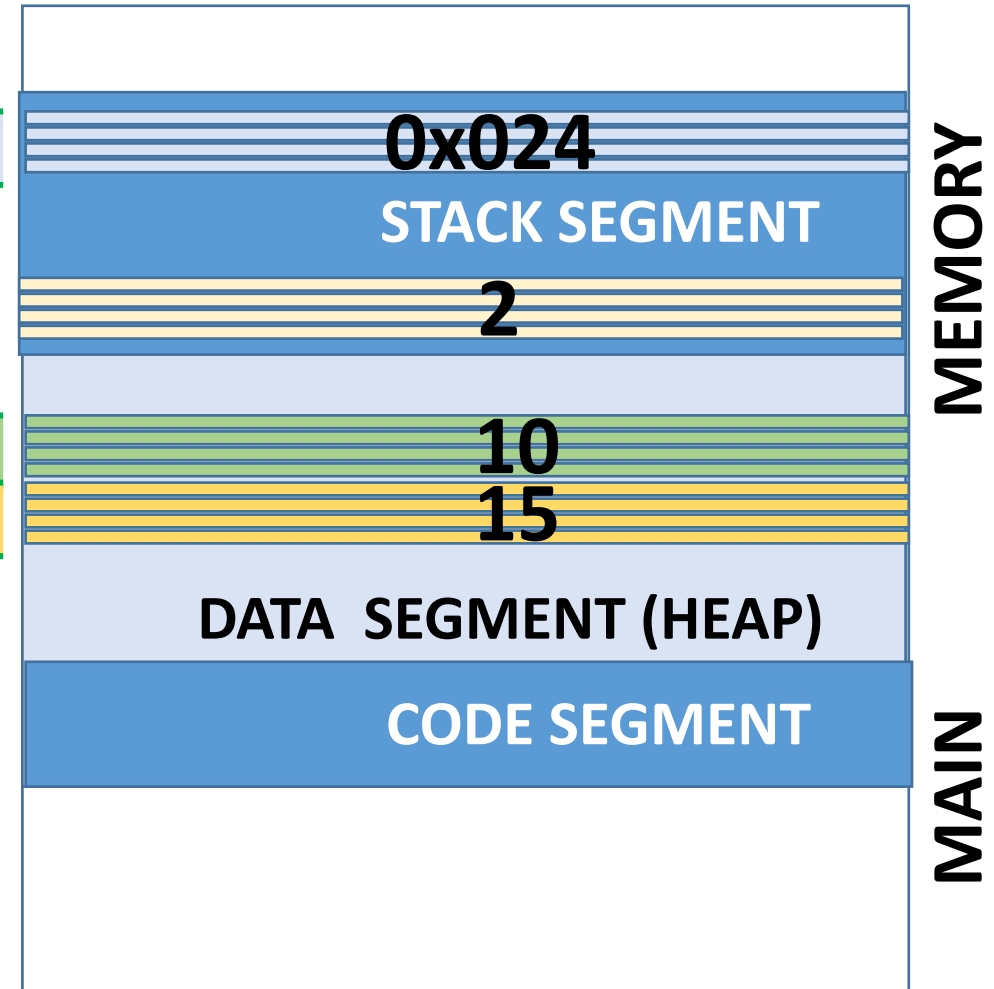
```
    A = new int[numStudents];
```

```
    A[0] = 10; A[1] = 15;
```

```
    return 0;
```

```
}
```

Address
0x024



Memory For Executing A Program (Process)

```
int main()
```

```
{
```

```
    int numStudents;
```

```
    int * A;
```

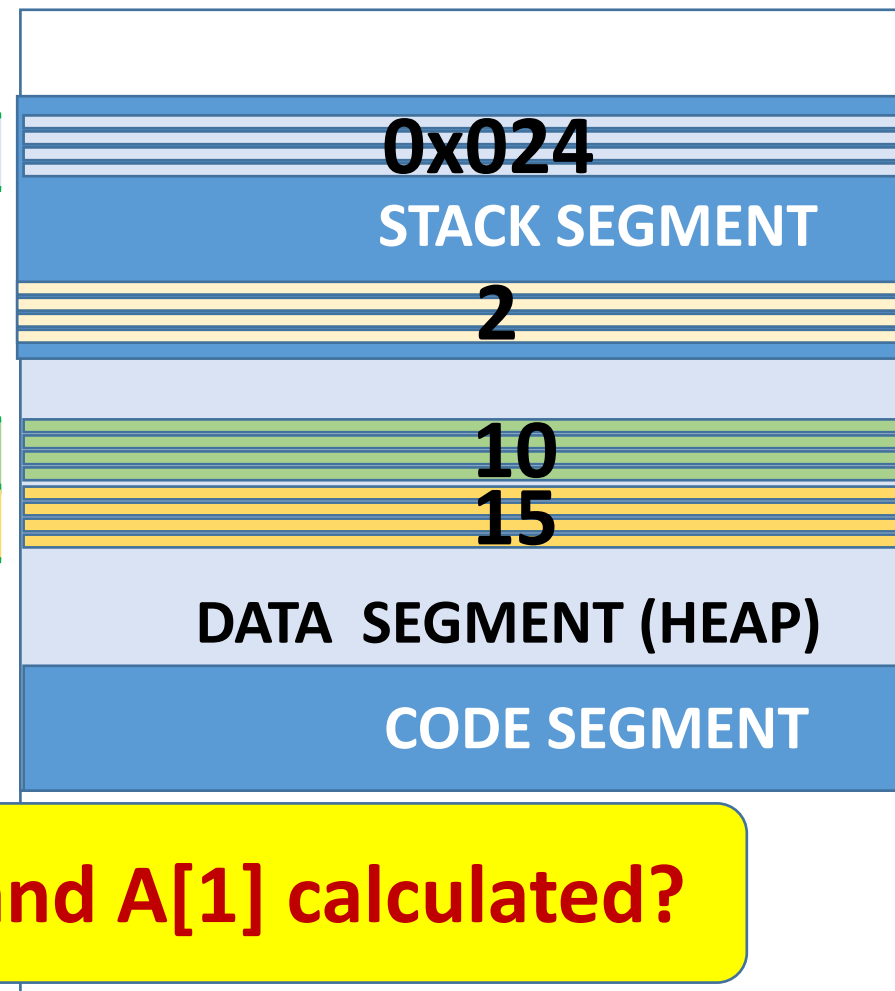
```
    cin >> numStudents;
```

```
    A = new int[numStudents];
```

```
    A[0] = 10; A[1] = 15;
```

```
    return 0;
```

Address
0x024



How are addresses of A[0] and A[1] calculated?

Calculating Addresses for Dynamic Arrays



- Compiler knows
 - A is pointer to an integer that is the first in an array of integers
 - How? From “**A = new int[numStudents];**”
 - Each element of the array is of **int** data type
 - Each int takes 4 consecutive memory locations (bytes)
- Therefore,
 - Address of A[0] is $(A + 0)$
 - Address of A[i] is $(A + (4*i))$

Address Arithmetic

Generic Format for Dynamic Memory Allocation



- To dynamically allocate memory for a variable of type T
T * myVarPtr;
myVarPtr = new T;
Variable accessed as ***myVarPtr**
- To dynamically allocate memory for an array of n elements of type T
T * myArray;
myArray = new T[n];
Array elements accessed as **myArray[0] ... myArray[n-1]**

Generic Format for Dynamic Memory Allocation

- To dynamically allocate memory for a variable of type T

`T * myVarPtr;`

`myVarPtr = new T;`

Variable accessed as ***myVarPtr**

**“new” returns
pointer to T
in both cases**

- To dynamically allocate memory for an array of n elements of type T

`T * myArray;`

`myArray = new T[n];`

Array elements accessed as **myArray[0] ... myArray[n-1]**

**Array of type T treated as a
variable of type T * that can be
indexed using [...]**

Good Programming Practices



- Most often, “new” will successfully allocate requested memory from heap and return address to the first allocated byte
- **However, we cannot take “new” for guaranteed**
 - **C++ allows “new” to fail and return 0x0 (also called NULL pointer)**
- **Always check if “new” has returned an address other than 0x0 (NULL) before dereferencing that address**
 - Avoids unnecessary program crashes especially if program dynamically allocates too much memory
 - This is real !!! Programmers encounter this situation in real-life

Dynamic Memory Allocation: The Right Way

```
int main()
{
    int numStudents;
    int * A;
    cin >> numStudents;
    A = new int[numStudents];
    if (A != NULL) { A[0] = 10; A[1] = 15;}
    return 0;
}
```

**Note the check for “new”
having succeeded**

Summary



- Dynamically allocating memory on heap (data segment)
- “new” construct in C++
- Accessing dynamically allocated variables and arrays
- Good programming practices when using dynamically allocated memory