

C++ for Java Programmers

C++ for Java Programmers

Lecture 6

1

C++ for Java Programmers

More pointing action

- Yesterday we considered:
 - Pointer Assignment
 - Dereferencing
 - Pointers to Pointers to Pointers
 - Pointers and Array
 - Pointer Arithmetic

2

C++ for Java Programmers

Today's Lecture

- And now we are going to look at:
 - The Stack and the Heap
 - Dynamic Memory Allocation
 - New and Delete
 - More Pointer Pitfalls
 - Putting it all together
- I discovered the transition button in powerpoint and got carried away.

3

C++ for Java Programmers

What do we know?

- We have learnt how to use the **dereferencing** and **referencing** operators to manipulate pointers and the data to which they point.
- We looked at how we can use pointer notation as alternative for manipulating arrays.
- We saw how pointers are the mechanism used by which C++ processes command line arguments.

4

C++ for Java Programmers

pointers are useful

- Consider the **strlen** exercise from the last lab sheet. We can solve this problem without using arrays at all...

```
int strlen(const *str)
{
    int i;
    while (*str++) i++;
    return i;
}
```

- So all in all we have seen that pointers can be handy things to have in your coding knowledge.


5

C++ for Java Programmers


Dynamic Memory

- But these roles are not the main reasons that pointers are part of the C++ language.
- The major role of pointers is the creation of dynamic structures.
- Particularly useful when you don't know beforehand exactly how much memory you need.
- Notably its through dynamic memory that the Standard Template Library (STL) can provide its functionality


6

C++ for Java Programmers 

SCOPE again...

- Almost all the variables, structures and objects we have used so far in our programs have generally been local objects.
 - They come into existence when their definition is executed.
 - They die when the block of code in which they live comes to an end. 
- The same fundamental rules apply to **global** variables too – they are created when defined, and destroyed when the program ends.


7

C++ for Java Programmers 

Not so for Dynamic Structures

- Dynamic objects are different in that they come in to memory due to direct memory allocation requests from the program.
- The dynamic objects continue to exist until the program returns their memory. But again it needs a direct command to do this.
- *The lifetime of a dynamic object is independent of its defining scope.*


8

C++ for Java Programmers 

Different Memory

- The memory comes for a dynamic structure comes from the free store.
- This is controlled by the OS, and is memory that can be broken up and allocated to a program as needed.
- Principal C++ way of acquiring this memory is using the **new** command.
- Deallocating it is handled by the **delete** command.


9

C++ for Java Programmers 

Quick mention of C-style

- Unless you have coded in C you might not appreciate the simplicity of **new** and **delete**.
- In C programming you use **malloc()**, **calloc()**, **realloc()**, and **free()** to dynamically allocate memory.
- OS such as windows can complicate things even further, offering its own functions.
- Although these things aren't amazingly difficult, C++'s **new** and **delete** can save you a lot of headaches.


10

C++ for Java Programmers 

Memory

- Memory, its properties and organization is a very simple idea to comprehend.
- One should keep in mind the simple idea that a computer is little more than a over-grown calculator with better I/O
- Memory is nothing more than a linear set of bytes. The computer needs a method to access all the memory on the system and it does this through addressing: giving each byte in memory an identity.

11

C++ for Java Programmers 

Segments

- Memory within a computer typically is made of *segments* - These segments are simply regions of memory with a starting and ending address, assigned attributes.
- A program cannot write to a segment that does not have the write permission attribute assigned to it.
- To do with common problems like General Protection Faults, Unrecoverable Application Errors, and such like.

12

C++ for Java Programmers

Segments

- When a program runs in memory, it has the following memory organization:

13

C++ for Java Programmers

The Stack

- In general, a stack is a single ended data structure with a first in, last out data ordering.
- The stack is useful for storing context - A function simply pushes all its local variables onto the stack when it enters, and pops them off when its done.
- Its complete context is nicely cleaned up afterwards.
- This is what happens with local variables.

14

C++ for Java Programmers

The Heap

- The heap is basically all the rest of memory the program has.
- Programs use the heap to hold data that must exist after a function call return.
- The heap holds values that have lifetimes more than local variables but less than global variables.
- To achieve this, the heap uses *memory allocation*. In this technique, all the memory in the heap is managed by a pair of algorithms which make reservations.

15

C++ for Java Programmers

The New operator

- The operator new has 3 forms.
- Simplest is a memory allocation request for a single object.
- This request expects the name of a type as the right operand.
- If there is sufficient unallocated free store memory, the operation returns a pointer to a memory location for that type.

16

C++ for Java Programmers

Example code

```
char *cptr;
cptr = new char;
```

17

C++ for Java Programmers

The data with no name

- Whatever type we are creating the specific amount of memory will be set up.
- Once a pointer has been successfully set with a location returned by a new request, the type at the location can be used like any other object of that type.
- In particular it can be evaluated and manipulated.
- However, note that this object has no name.

18

C++ for Java Programmers

What if we lose it?

- So variable names are fixed – but our pointer is no constant. We might change it by accident.
- So how do we get access to our nameless data again?
- We can't. It is gone. adios. forever. wave goodbye... 🤖
- *Even worse now* we can't delete it and its just going to hang around, clogging up memory until we do something drastic.

19

C++ for Java Programmers

Example code

```
char* cptr;
cptr = new char;
cin >> *cptr;
```

We dereference the pointer and so access the dynamic memory space we have been allocated

stack heap

20

C++ for Java Programmers

Second form of new

- A second form of the new operator not only requests a single dynamic object, but initialises it too.
- The initialiser values are just specified within parentheses, using commas to separate any initialisers.
- So in general form:

```
pointer = new type(initialisers);
```

21

C++ for Java Programmers

Example code

```
char* mike;
int* nilgun;
mike = new char('A');
nilgun = new int(1);
```

Here we initialise the nameless data.

stack heap

22

C++ for Java Programmers

Third form of new

- The third form permits multiple dynamic objects to be requested in a single operation.
- This is like creating an array of objects in dynamic memory.
- The form expects a type name along with a subscripted expression indicating the number of requested objects of that type.
- If there's enough contiguous and unallocated free memory a pointer is returned. This contains the memory address of the heap space we have been given.

23

C++ for Java Programmers

Example code

```
float* numbers;
numbers = new float[4];
```

Here we initialise the nameless data.

stack heap

24

C++ for Java Programmers

The Point of pointers

- So again how is this any better than using an array created on the stack? Why not just use a normal local variable.
- The point is that although we used a constant expression in our request for multiple dynamic objects we *didn't have to*.
- You can use **variables, or expressions**.
- That means that we can create structures like arrays in dynamic memory, and determine their size during runtime. We don't have to know the size of the array.

25

C++ for Java Programmers

Example code

```
cout << "Go on. pick an array size" << flush;
int listSize;
cin >> listSize;
int* values = new int[listSize];
```

Here we combine two statements.

We use a variable with the new command

26

C++ for Java Programmers

Yin and Yang - Delete

- Everything that you create with the new command you **MUST** delete too.
- This often turns out to be more responsibility than some budding programmers can cope with.
- Using the delete operation we can free up that dynamic memory in the heap.
- But we need that pointer to tell it which memory location to free up.

27

C++ for Java Programmers

Example code

```
char* mike;
int* nil;
mike = new char('A');
nilgun = new int(1);
.....
delete mike;
delete nilgun;
```

28

C++ for Java Programmers

Take care with your pointers

- What happens to the pointers we used?
- **Nothing** – the value of the pointer just becomes undefined.
- It is good form because of this to set the pointer to null after you have used it in a delete statement;

```
.....
delete mike;
mike = null;
delete nilgun;
nilgun = null;
```

29

C++ for Java Programmers

Dangling Pointers

- You have to be really careful of this as a programmer, especially if your code contains multiple pointers to the same object.

```
char *ptr1;
char *ptr2;

ptr1 = new char('C');
ptr2 = ptr1;

delete ptr1;
cout << *ptr2;
```

30