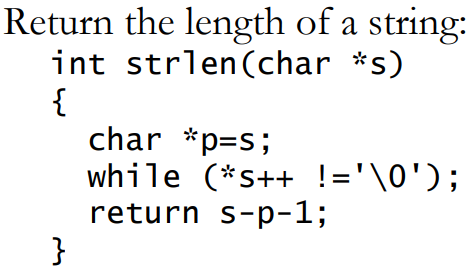
Computer Systems Lecture 9

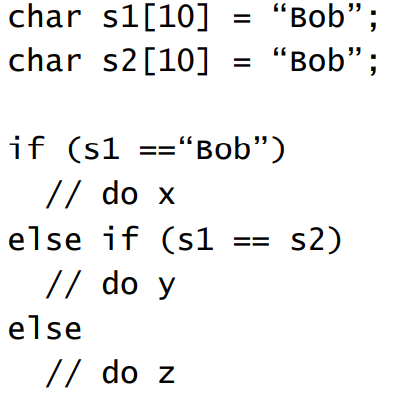
More Pointer Arithmetic

Common expressions:

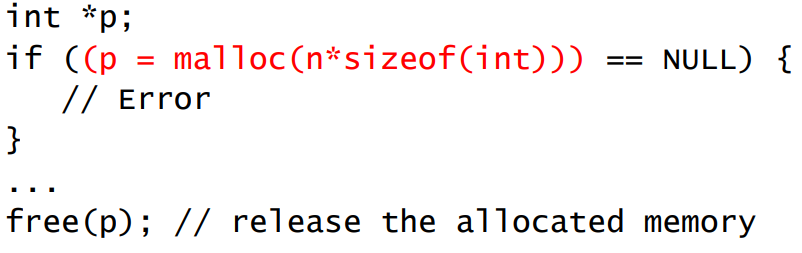
* \*p++ this retrieves the value pointed to by p and then moves p to point to the next element
* \*++p this moves p to point to the next element then retrieves that element.
* (\*p)++ increments the value at p
* P=NULL is used to make p point to nothing (!p) will return true if the pointer points to nothing.
* Dereferencing a null pointer will cause a segmentation fault.



S is local, so we can change it. The pointer is incremented, dereferenced and compared all in one, no body for the loop is needed. At the end of the loop s points to the end of the string, we return s-p (p is the start of the string) -1 to get the number of characters between where we were placed in the array and the null character.



Here z will run as s1 is a pointer which will not point to the literal string “Bob” so the first condition fails, the second condition also fails as the two arrays will be held at different places in memory.

Dynamic Memory Allocation  
Pointers aren’t much use with statically allocated data. We can use malloc to allocate a chunk of memory at run time and return the address:

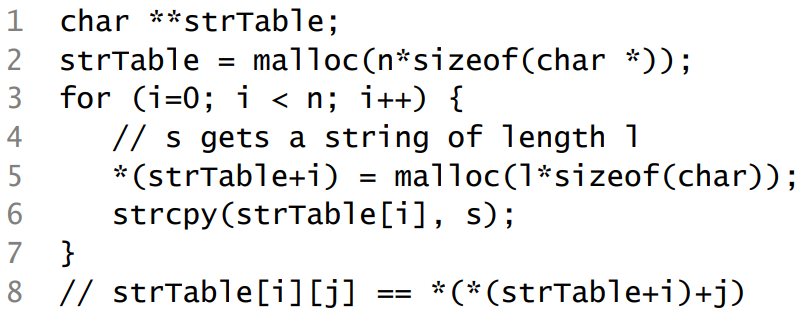
Note the free(p) at the end, mallocs must always have a corresponding free.

Pointers to Pointers

Consider an array of strings char \*strTable[10];

The strings are dynamically allocated and so can be any size, but the table size is fixed to 10 strings. If we don’t know the number of strings ahead of time we’ll need to dynamically allocate the storage for the array of strings as well, meaning we should define: char \*\*strTable;

Space must be allocated for both the table and the strings themselves:



Memory Regions and Management

Memory Areas

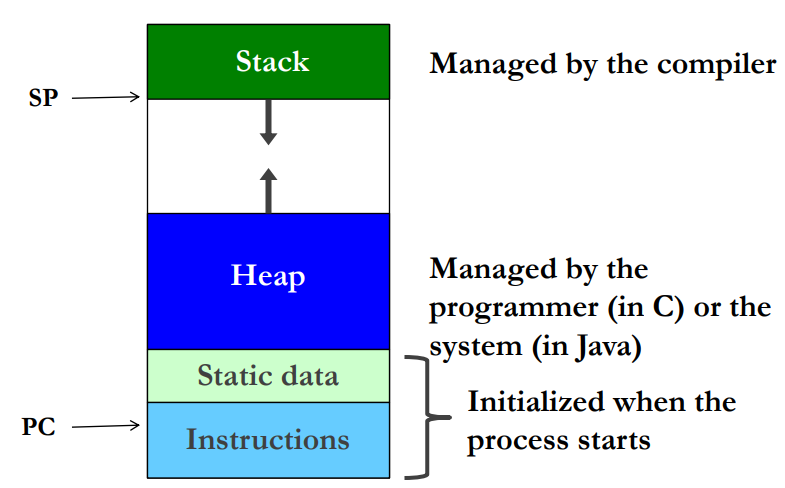
* Heap – dynamically allocated storage
* Stack – for function/method local variables
* Static – for data live during the entire program lifetime

In Java

* All objects on heap
* Unusable objects on heap are recycled automatically by garbage collection

In C

* Data structures in all 3 areas
* Programs must explicitly free-up heap storage that is no longer needed.



Categories of Variables in C

Global Variables (statically allocated)

* Defined outside of functions
* Have lifetime of program and scope to file end
* Extern declarations extend scope before definition and to other files
* Declare static to hide from other files

Local (automatic) variables (allocated on stack)

* Defined inside a function
* Not available outside function
* Distinct storage for each function invocation
* Declare static for same storage for all invocations

Compilation Units

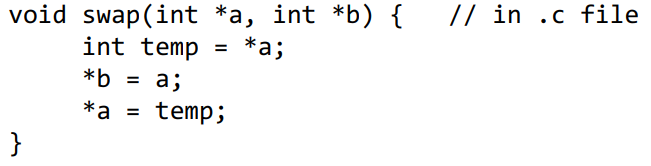
* Programs are divided into compilation units
  + Provides degree of modularity
  + Each commonly has a main file (.c) for source code
  + Header files (.h) declare public interfaces of units
* Each compiled separately to relocatable object code
  + Allows creation of object-code libraries
* A linker assembles these into an executable, resolving references between units
* A loader sets up the executable program in memory and initialises data areas, prior to the program being run
  + Loader also computes addresses for jump instructions

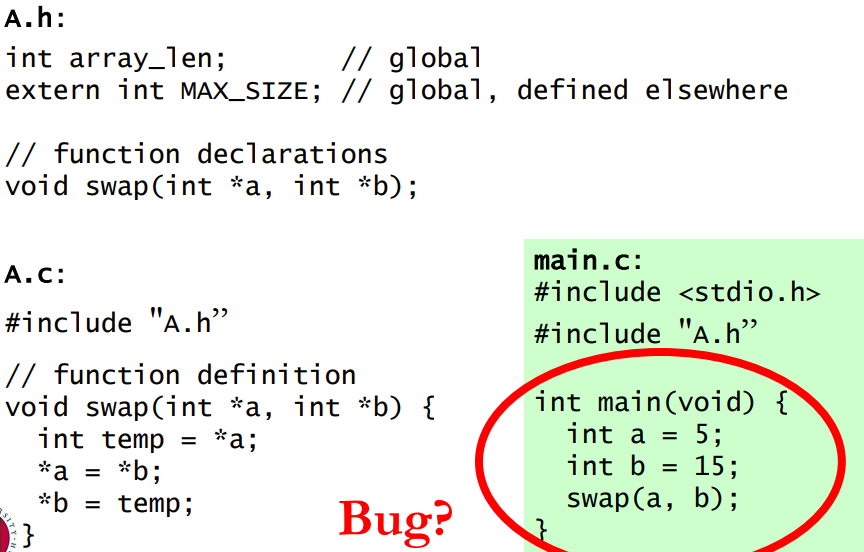
Declaration vs Definition

Declaration: inform the compiler of the existence of a variable or function



Definition: provide function body; allocate memory for globals

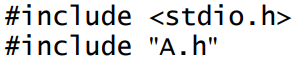




The C pre-processor: cpp

The C pre-processor deals with:

* Includes – imports header files



* Text substitution, e.g. defining constants



* Macros (inline functions)



* Conditional compiliation

