Individual Exercises - Lecture 17

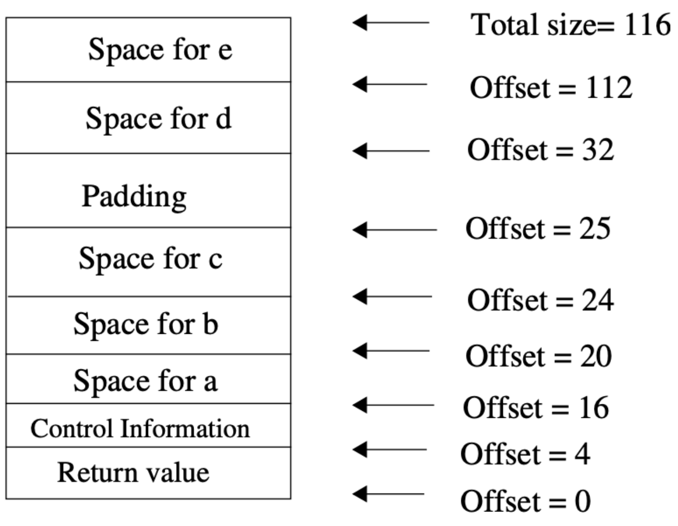
1. Read the articles under additional references
   Can be found here:
   http://www3.nd.edu/~dthain/courses/cse40243/spring2006/gc-survey.pdf
   www.memorymanagement.org
   https://www.memorymanagement.org/mmref/begin.html
   https://www.c-sharpcorner.com/article/memory-management-in-net/
   https://www.artima.com/insidejvm/applets/HeapOfFish.html
   https://www.oracle.com/technetwork/java/javase/tech/memorymanagement-whitepaper-1-150020.pdf

2. Do Fischer et al exercise 1, 3, 4, 5, 9, 12, 15, 20 on pages 482- 488 (exercise 2, 3, 4, 6, 11, 13, 16, 19 on pages 514-520 in GE)

   1. Show the frame layout corresponding to the following C function:

```c
int f(int a, char *b) {
   char c;
   double d[10];
   float e;
   ...
}
```

   Assume control information requires 3 words and that f's return value is left on the stack. Be sure to show the offset of each local variable in the frame and be sure to provide for proper alignment (integers and floats on word boundaries and doubles on doubleword boundaries)



Note that there are seven bytes of padding following variable c because the array of doubles, d, requires double word alignment. The size of the frame could be reduced by placing the float e *before* d. Such reordering is rarely done because memory space currently is very cheap and programmers sometimes want to be able to anticipate where variables will be placed (for debugging and cache optimization).

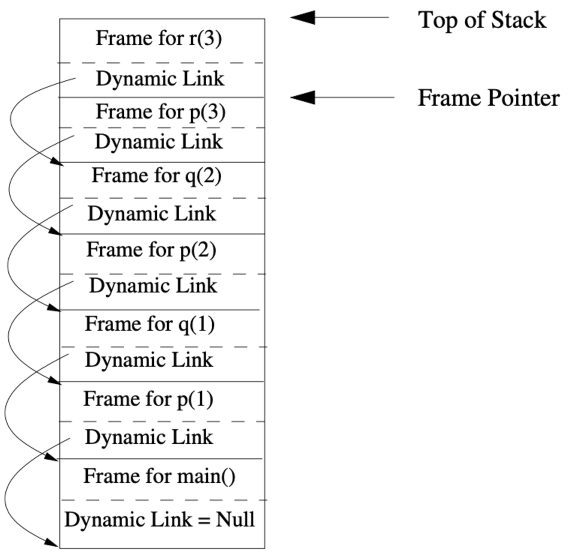
On the other hand, being able to predict where variables will be placed in a frame makes it far easier to violate security rules. Hackers often access vital runtime data (like return address or heap pointers) by overwriting valid values from adjacent fields. Making data layout in a frame unpredictable enhances overall security.

3. Using the code below, show the sequence of frames, with dynamic links, on the stack when r(3) is executed assuming we start execution (as usual) with a call to main().

```
r(flag){
   printf("Here !!!\n"); }
q(flag){
   p(flag+1); }
p(int flag){
   switch(flag){
      case 1: q(flag);
      case 2: q(flag);
      case 3: r(flag); }

main(){
   p(1);
}
```

The frames existant when r(3) executes is shown below:

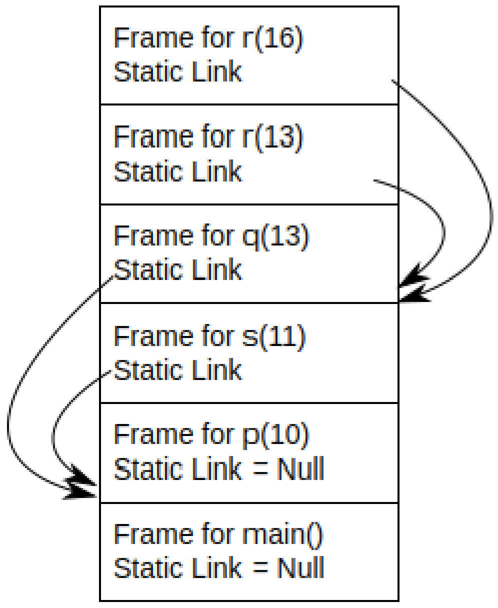


4. Consider the following C-like program that allows subprograms to nest. Show the sequence of frames, with static links, on the stack when r(16) is executed assuming we start execution (as usual) with a call to main(). Explain how the values of a, b, and c are accessed in r's print statement.

```
p(int a){
   q(int b){
      r(int c){
         print(a+b+c);
      }
      r(b+3);
   }

   s(int d){
      q(d+2);
   }

   s(a+1);
}
main(){
   p(10);
}
```



The frame for r(16) has access to 'c' through itself. Access to 'b' can be obtained by following the static link to the frame of q(13). 'a' can be found by following static links to frame q(13) and from there to the frame of p(10).

5. Reconsider the C-like program shown in Exercise 4, this time assuming display registers are used to access frames (rather than static links). Explain how the values of a, b, and c are accessed in r's print statement.
Display registers are described in section 12.2.4. For this sequence of calls we would need a total of 3 display registers(because the deepest nesting level is three).

Again, accessing c is trivial as it is stored in the display of r(16) (d2).
Looking at d1 we get a reference to the display of method call q(13) which gives us access to the 'b' variable. Finally we look at d0, which references the display of the method call p(10). Here we have access to the 'a' variable.

9. Assume that in C we have the declaration int a[5][10][20], where a is allocated at address 1000. What is the address of a[i][j][k] assuming a is allocated in row-major order? What is the address of a[i][j][k] assuming a is allocated in column-major order?
Row major order:
Given a cube with the following dimension: i_length, j_length, k_length, then the offset of a given index can be calculated as the following:
$OffsetRowMajor(i, j, k) \ = \ i * j_{length} * k_{length} + j * k_{length} + k \ = (i * j_{length} + \ j) * k_{length} + k$

For this example we have that i_length = 5, j_length = 10, k_length = 20, then the offset of the index 2, 1, 2 (i = 2, j = 1, k = 2) can be calculated in the following way:
$Offset(2, 1, 2) \ = (2 * 10 + \ 1) * 20 + 2 \ = 422$
$MemoryAddressRowMajor(2, 1, 2) \ = \ 1000 \ + \ OffsetRowMajor(2, 1, 2) \ = \ 1422$

Column major order:
The offset using column major order can be calculated in the following way:
$OffsetColMajor(i, j, k) \ = i + j * i_{length} + k * i_{length} * j_{length}$

Using the same example as before (2,1,2 or i = 2, j = 1, k = 2) we get the following allocation address:
$OffsetColMajor(2, 1, 2) \ = 2 + 1 * 5 + 2 * 5 * 10 \ = \ 107$
$MemoryAddressColMajor(2, 1, 2) \ = \ 1000 \ + \ OffsetColMajor(2, 1, 2) \ = \ 1107$

For an in-depth explanation of row-major and column-major order for multi-dimensional arrays see:
https://eli.thegreenplace.net/2015/memory-layout-of-multi-dimensional-arrays

12. Assume we add a new option to C++ arrays that are heap-allocated, the flex option. A flex array is automatically expanded in size if an index beyond the array's current upper limit is accessed. Thus we might see:

```
ar = new flex int[10];
ar[20] = 10;
```

The assignment to position 20 in ar forces an expansion of ar's heap allocation. Explain what changes would be needed in array accessing to implement flex arrays. What should happen if an array position beyond an array's current upper limit is read rather than written?
As the amount of memory currently allocated for the array can only hold 10 values, a new block of memory must be allocated from the heap. Then all elements from the original array must be copied to the new array and the new element 10 written to location 20. When allocating a block of memory for the new array it is often better to grow the array with a certain factor (e.g., 2x) instead of just making room for the new elements to reduce the number of times the array must be resized. In Java ArrayList implements an array that dynamically resizes when elements are added, the implementation used for OpenJDK 8 can be found at [1].

The semantics for reading values from outside the bounds of a flex array is much less clear. Two safe options would be to return a default value for the type stored in the array (e.g., return 0 for int) or raise an error to inform the programmer that the index provided is outside the bounds of the array. An unsafe options would be to resize the array and then return the "garbage" value at the provided index (if all values are initialized this is not a problem).

[1]
http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/tip/src/share/classes/java/util/ArrayList.java

15. Assume we organize a heap using reference counts. What operations must be done when a pointer to a heap object is assigned? What operations must be done when a scope is opened and closed?

When doing an assignment, the reference count of the heap reference being overwritten is decreased by one and the reference count of the heap reference being assigned is increased by one. Source and target locations containing null are treated as special cases.

When a scope is opened, all heap references not explicitly initialized are set to null or some special "error value." When a scope is closed, all heap references that are lost are treated as if they had been assigned null (thereby decreasing reference counts to account for the lost heap references).

20. The second phase of a mark-sweep garbage collector is the sweep phase, in which all unmarked heap objects are returned to the free space list. Detail the actions needed to step through the heap, examining each object and identifying those that have not been marked (and hence are garbage).

Assume that the first word of each object is used to hold the length of the object and the sign bit is used for marking. Then returning objects to the free list is a simple run through the heap from its start address looking at the sign bit to determine if the objects should go in the list. Next object is found by adding the length and the current object to the address of the current object.

3. Do Sebesta exercise 7 and 8 page 468

   7. It is stated in this chapter that when nonlocal variables are accessed in a dynamic-scoped language using the dynamic chain, variable names must be stored in the activation records with the values. If this were actually done, every nonlocal access would require a sequence of costly string comparisons on names. Design an alternative to these string comparisons that would be faster.
   Use string mapping to replace variable names with integer placeholders that can be accessed in a linear look up time, with a hashtable. This increases performance, since integer comparison is cheap.

   8. Pascal allows gotos with nonlocal targets. How could such statements be handled if static chains were used for nonlocal variable access? Hint: Consider the way the correct activation record instance of the static parent of a newly enacted procedure is found (see Section 10.4.2).

   Following the hint stated with the question, the target of every goto in a program could be represented as an address and a nesting_depth, where the nesting_depth is the difference between the nesting level of the procedure that contains the goto and that of the procedure containing the target. Then, when a goto is executed, the static chain is followed by the number of links indicated in the nesting_depth of the goto target. The stack top pointer is reset to the top of the activation record at the end of the chain.

Group Exercises - Lecture 17

1. Discuss the outcome of the individual exercises
   Did you all agree on the answers?

2. Do Fischer et al exercise 2, 7, 11, 10, 13, 16, 19, 25, 24 on pages 482-488 (exercise 1, 8, 9, 10, 12, 15, 18, 24, 25 on pages 514-520 in GE)

   2. Local variables are normally allocated within a frame, providing for automatic allocation and deallocation when a frame is pushed and popped. Under what circumstance must a local variable be dynamically allocated? Are there any advantages to allocating a local variable statically (i.e., giving it a single fixed address)? Under what circumstances is static allocation for a local permissible?

   The variable must be dynamically allocated whenever the corresponding function/method/procedure can be activated more than once, i.e. there is more than one activation record for the function/method/procedure on the stack (think e.g. of a recursive function).

   Some languages allow the programmer to explicitly specify that a local function variable should be static, which causes the variable to persist across all calls of the function.

   Local constants can also be statically allocated.

   7. Although the first release of Java did not allow classes to nest, subsequent releases did. This introduced problems of nested access to objects, similar to those found when subprograms are allowed to nest. Consider the following Java class definition:

```java
class Test {
   class Local {
      int b;
      int v(){ return a+b; }
      Local(int val){ b=val; }
   }

   int a = 456;

   void m(){
      Local temp = new Local(123);
      int c = temp.v();
   }
}
```

   Note that method v() of class Local has access to field a of class Test

as well as field b of class Local. However, when temp.v() is called, it
is given a direct reference only to temp. Suggest a variant of static links
that can be used to implement nested classes so that access to all visible
objects is provided.

The problem is that the frame produced from the call to .v() has only a static reference to temp (the instance of the 'Local' class). From this we have no reference to variable 'a' from the test class instance.

However, nested class instances can only exist if their outer class is also instantiated. So to solve our problem: when creating an instance of the nested class(using the new keyword), we simply include a pointer(static link) to the outer class.
Now when .v() is called, it will contain a static link to the instance of the Local class, which then again contains a static link to the 'Test' class instance.

11. In Java, subscript validity checking is mandatory. Explain what changes would be needed in C or C++ (your choice) to implement subscript validity checking. Be sure to address the fact that pointers are routinely used to access array elements. Thus you should be able to check array accesses that are done through pointers, including pointers that have been incremented or decremented.

This would require us to know how an array is represented in memory and being able to check that a pointer still points to the array elements, e.g. by going from the current element back to the start of the array, where the size is kept.

First, arrays in C should be made first-class types that consist of a pointer and a size. This would allow arrays to be both passed to/from functions and have their bounds checked as their size is always available. Both explicit and implicit coercion of arrays to pointers should not be allowed. Pointer arithmetic should also be disallowed in order to prevent data from being accessed outside the bounds of an array. However, many existing programs would have to be updated to account for these changes. Adding bounds-checking to C has been a focus for much research with approaches suggested that are backwards compatible [1] and some that extend C [2].

[1] Dinakar Dhurjati and Vikram Adve, Backwards-compatible array bounds checking for C with very low overhead, https://dl.acm.org/doi/10.1145/1134285.1134309

[2] Andrew Ruef, Leonidas Lampropoulos, Ian Sweet, David Tarditi, and Michael Hicks, Achieving Safety Incrementally with Checked C, https://rd.springer.com/chapter/10.1007/978-3-030-17138-4_4

10. Most programming languages (including Pascal, Ada, C, and C++) allocate global aggregates (records, arrays, structs, and classes) statically, while local aggregates are allocated within a frame. Java, on the other hand, allocates all aggregates in the heap. Access to them is via object references allocated statically or within a frame. Is it less efficient to access an aggregate in Java because of its mandatory heap allocation? Are there any advantages to forcing all aggregates to be uniformly allocated in the heap?

Access to a heap object requires a heap validity check and an indirection. If the object is accessed frequently, these overheads can probably be optimized away (as redundant or loop-invariant operations). If the heap manager does not compact the heap, a program may find active objects spread among inactive heap allocations, increasing the program's "footprint" in memory.

An advantage of allocating all aggregates in the heap is that the size of the aggregate can be part of its value rather than its declaration. Java arrays can change their size during execution. This can be very useful when no fixed size bounds are apparent (e.g., a character buffer being edited, or an array being sorted). On the other hand, sometimes it is useful to fix the size of an array and allocate it as efficiently as possible (e.g., an array representing a chess board). C# has added fixed-size arrays (that may be allocated statically or within a frame) as a language extension. (Java-style arrays are of course retained.)

13. Fortran library subprograms are often called from other programming languages. Fortran assumes that multidimensional arrays are stored in column-major order; most other languages assume row-major order. What must be done if a C program (which uses row-major order) passes a multidimensional array to a Fortran subprogram. What if a Java method, which stores multidimensional arrays as arrays of array object references, passes such an array to a Fortran subprogram?

When calling a Fortran library from C the programmer must manually reorganize the multidimensional array so the values are stored in column-major as expected by Fortran. When combining C and Fortran code it is recommended to perform all of the array operations in either C or Fortran to remove the problem. When calling Fortran from Java a multidimensional array in column-major order must be manually constructed where the references are replaced with the actual data and passed it to Fortran.

16. Some languages, including C and C++, contain an operation that creates a pointer to a data object. That is, p = &x takes the address of object x, whose type is t, and assigns it to p, whose type is t*.
How is management of the runtime stack complicated if it is possible to create pointers to arbitrary data objects in frames? What restrictions on the creation and copying of pointers to data objects suffice to guarantee the integrity of the runtime stack?

Creating a pointer to a location in a frame is very dangerous as the pointer may be used *after* the frame is popped from the stack. Hence modern successors to C and C++, like Java and C♯, disallow this operation.

There are two approaches to handling pointers to frame locations. We may add the rule that a pointer to a frame location may never be assigned to a variable with a longer lifetime than the frame itself. This is a form of *escape analysis* that tracks where a frame pointer may be assigned.

An alternative, taken by some functional languages like ML, is to allocate frames in the heap rather than on a stack. Now the rules of garbage collection apply; when no pointers to a frame remain, the frame is collected. As long as a pointer remains, the frame is protected from garbage collection. It may appear that using the heap to hold frames is needlessly inefficient, but this need not be the case. Modern *copying collectors* (Section 12.4.3 on page 472) have the property that their cost is controlled by the number of *live* heap objects. Dead objects, like frames for completed calls, are essentially “free.”

19. In a strongly typed language such as Java, all variables and fields have a fixed type known at compile-time. What runtime data structures are needed in Java to implement the mark phase of a mark-sweep garbage collector in which all accessible (“live”) heap objects are marked?

We must trace all pointers (references) found in global, stack and heap space. For global pointers we can simply have a list of global addresses known (from their declarations) to hold heap pointers. To save space, a *bit map* might be used. In a bit map, each bit represents one word of global data space. A one bit means the word is a pointer and a zero says it is not.

To find pointers within a frame, we start with the current frame pointer. Each frame, in its control information, will contain a method code identifying the method the frame corresponds to. Each method code will index a list (or bitmap) of frame offsets known to hold heap pointers.

Each heap allocation has a header word. This header contains a type code identifying the class the heap object implements. We use this type code to index a list of offsets within the object that contain pointers. For array objects that contain pointers, we can extract the size of the array and where within the object the array elements begin.

25. An unattractive aspect of both mark-sweep and copying garbage collection is that they are batch-oriented. That is, they assume that periodically a computation can be stopped while garbage is identified and collected. In interactive or real-time programs, pauses can be quite undesirable. An attractive alternative is concurrent garbage collection in which a garbage collection process runs concurrently with a program.

Consider both mark-sweep and copying garbage collectors. What phases of each can be run concurrently while a program is executing (that is, while the program is changing pointers and allocating heap objects)? What changes to garbage collection algorithms can facilitate concurrent garbage collection?

Mark-sweep: The Mark sweep method does not immediately allow for concurrent garbage collection. The releasing of heapvars to the freelist could, however, possibly be done concurrently in the following manner. First halt the program, mark all vars without any references. Then resume execution. Now a concurrent process could run, that cleans the marked variables and puts them back into the free list.

Copying garbage collectors: The problem with running the copying GC concurrently, is the possibility of copying memory that is being modified by the running program. Modifications to the GC algorithm are therefore required in order to safely run concurrently. Take a look at the additional references below.

For more information on GC in real time systems, take a look at:

https://en.wikipedia.org/wiki/Tracing_garbage_collection#Tri-color_marking

https://www.drdobbs.com/jvm/java-garbage-collection-for-real-time-sy/184410684

https://www.ibm.com/developerworks/library/j-rtj4/

24. Copying garbage collection can be improved by identifying long-lived heap objects and allocating them in an area of the heap that is not collected.

What compile-time analyses can be done to identify heap objects that will be long lived? At runtime, how can we efficiently estimate the “age” of a heap object (so that long-lived heap objects can be specially treated)

Compile Time: Possibly follow references from the Main class and somehow estimate how long each reference lives. For example, objects allocated and stored on the Main object will probably live for the majority of the program.

Runtime: copying garbage collectors can be extended with generations. Objects are allocated in the youngest generation and after they have survived garbage collection *n* times they are moved to the next generation. Later generations are garbage collected less frequently, and the oldest generation might not even be garbage collected. For more information see Generational Garbage Collection in Fisher Section 12.4.

3. What are the design issues for pointer types?
   The primary design issues particular to pointers are the following:
   1. What are the scope and lifetime of a pointer variable?
   2. What is the lifetime of a heap-dynamic variable (the value a pointer references)?
   3. Are pointers restricted as to the type of value to which they can point?
   4. Are pointers used for dynamic storage management, indirect addressing, or both?
   5. Should the language support pointer types, reference types, or both?

   For more detail see Sebesta Section 6.11

4. What are the two most common problems with pointers?
   Dangling Pointer: a pointer that is pointing to a location in memory that has been deallocated, using this pointer is dangerous as the memory might have been allocated for some other data, possibly even another data type.

   Lost Heap-Dynamic Variable: if data is allocated on the heap but the pointer to the data is lost, the program leaks memory as the data cannot be deallocated by the program.

   Also, Reading from uninitialized pointers provides either “garbage” values or causes a segmentation fault. Use of pointers can also make the program harder to read and comprehend.

   For more detail see Sebesta Section 6.11

5. Why are pointers in most languages restricted to pointing to a single type variable?
   A pointer, points to a specific place in memory. When dereferencing the pointer to get the value, the compiler needs to know how many bytes to read from the starting address. Some types have different lengths i.e 32 and 64 bit integers. So restricting a pointer to a single type ensures that the correct memory amount of memory is read. Consider if there were no types in the declaration of a pointer in C. The compiler would be able to perform some type inference for some simple programs, however, it would not be able to detect all types behind a pointer. The compiler would therefore not know how much memory to return. A solution to this would be to force the programmer to decorate all dereferencing of pointers with a type or number of bytes to read but that would introduce a lot of other potential problems.

6. Sebesta review questions 35, 36 page 321
   35. Why are reference variables in C++ better than pointers for formal parameters?

   Reference variables used as formal parameters in C++ always reference a value and are implicitly dereferenced unlike pointers. Resulting in safer and more readable code.

   See also:
   https://en.wikipedia.org/wiki/Reference_(C%2B%2B)#Relationship_to_pointers

   36. What advantages do Java and C# reference type variables have over the pointers in other languages?

   Java class instances are referenced with reference variables. Since Java instances are implicitly deallocated, there cannot be dangling references in Java. C# follows a similar design, however it includes pointers when using the unsafe keyword. If a object is pointed to by a pointer it will not be implicitly deallocated in C#

7. Sebesta problem set 13, 14 page 323 (in 14 discuss rather than write)
   13. Write a short discussion of what was lost and what was gained in Java's designers' decision to not include the pointers of C++.

   Compared to C++, Java provides a more limited interface for reading from and writing to memory by not including pointers, e.g., you cannot read data from a Java array of longs at an arbitrary byte offset. As such, the lack of pointers limits what can be expressed in Java to a small degree, on the other hand, it is much more difficult for the programmer to make a mistake when reading and writing to memory. In summary, it is a tradeoff between expressibility (e.g, allowing optimizations) and safety.

   14. What are the arguments for and against Java's implicit heap storage recovery, when compared with the explicit heap storage recovery required in C++? Consider real-time systems.

   Like stated for Problem Set 13, Java prioriotizes memory safety to a higher degree than C++ and therefore uses garbage collection to manage heap memory while C++ requires that the programmer have to manually deallocate memory through RAII (Resource Acquisition Is Initialization) and smart pointers like std::unique_ptr and std::shared_ptr (reference counting). By automating memory management a large collection of memory related bugs can be removed, e.g, Microsoft have stated that ~70% of the vulnerabilities they assign a CVE (Common vulnerabilities and Exposures) each year are memory safety issues. However, garbage collection comes at a cost as the programmer has no control over when it runs. This uncertainty is a problem for real-time systems where guarantees on execution time is required, e.g., the embedded system controlling a car's airbag. However, research into using Java for such applications has been performed [2].

[1]
https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code/

[2]
https://vbn.aau.dk/da/publications/hvmtp-a-time-predictable-and-portable-java-virtual-machine-for-ha

8. What is the expected level of use of pointers in C#? How often have you used pointers in C#?
   Raw pointers are almost never used in C#, e.g., they might be used for interfacing with existing C or C++ code, however, references to classes are pointers in themselves.