CMSC 330: Organization of Programming Languages

Garbage Collection

Memory attributes

- Memory to store data in programming languages has several attributes:
  - Persistence (or lifetime) – How long the memory exists
  - Allocation – When the memory is available for use
  - Recovery – When the system recovers the memory for reuse
- Most programming languages are concerned with some subset of the following 4 memory classes:
  - Fixed (or static) memory
  - Automatic memory
  - Programmer allocated memory
  - Persistent memory

Memory classes

- Static memory – Usually a fixed address in memory
  - Persistence – Lifetime of execution of program
  - Allocation – By compiler for entire execution
  - Recovery – By system when program terminates
- Automatic memory – Usually on a stack
  - Persistence – Lifetime of method using that data
  - Allocation – When method is invoked
  - Recovery – When method terminates

Memory classes

- Allocated memory – Usually memory on a heap
  - Persistence – As long as memory is needed
  - Allocation – Explicitly by programmer
  - Recovery – Either by programmer or automatically (when possible and depends upon language)
- Persistent memory – Usually the file system
  - Persistence – Multiple execution of a program (e.g., files or databases)
  - Allocation – By program or user, often outside of program execution
  - Recovery – When data no longer needed
  - This form of memory usually outside of programming language course and part of database area (e.g., CMSC 424)

Memory Management in C

- Local variables live on the stack
  - Allocated at function invocation time
  - Deallocated when function returns
  - Storage space reused after function returns
- Space on the heap allocated with malloc()
  - Must be explicitly freed with free()
  - This is called *explicit* or *manual* memory management
    - Deletions must be done by the user

Memory Management Mistakes

- May forget to free memory (*memory leak*)

```
{ int *x = (int *) malloc(sizeof(int)); }
```

- May retain ptr to freed memory (*dangling pointer*)

```
{ int *x = ...malloc();
  free(x);
  *x = 5; /* oops! */
}
```

- May try to free something twice

```
{ int *x = ...malloc(); free(x); free(x); }
```

  - This may corrupt the memory management data structures
    - E.g., the memory allocator maintains a *free list* of space on the heap that's available