Today’s lecture

- **Generics, void***
  - Library functions that operate on raw memory
  - Manual pointer arithmetic

- **Function pointers**
  - How client callback function coordinates with generic operation

- **C(pointer wrap)**
  - C as thin layer on underlying memory reality
Generic pointers

**void** `*` **pointer**
Variable of type address with unspecified/unknown pointee type

**What you can do with a void** `*`
Pass to/from function, pointer assignment

**What you cannot**
Cannot dereference
Cannot do pointer arithmetic
Cannot use array indexing (depends on both arithmetic & dereference!)

**Why do you want one?**
What gain is there in "forgetting" the pointer type?

**Generic functions!**
All data has an address, by referring to it by address we can smooth over differences in data type
Operations that rely on data type

- Process raw bytes without knowing what they are
  - memcpy, memmove, memchr, memcmp, ...

- What if need more meaningful behavior per-type?
  - Do "something" to each element or filter/sort elements

- Coordinate with client via callback function
  - Generic operation "calls back" to client who knows the specifics of data

- Generic implementation
  - Works in terms of void*, manual pointer arithmetic, raw memory operations
  - No knowledge of what the data is, only its size
  - Client refers to data by address (void * "forgets" knowledge of type)

- Generic client
  - Supplies the callback function that operates in specific on the data
  - Must cast void* back to specific type and dereference (cast "restores" knowledge of type)
Let’s code & draw!

/afs/ir/class/cs107/samples/lect9
generic.c
**Common void* idioms**

- **Call generic function**
  ```c
  qsort(arr, n, sizeof(arr[0]), compare_widget);
  bsearch(&key, arr, n, sizeof(arr[0]), compare_widget);
  ```

- **Comparison callback function**
  ```c
  int compare_widget(const void *a, const void *b)
  {
    const widget *first = (const widget *)a;
    const widget *second = (const widget *)b;
    ...
  }
  ```

- **Calculate address of the ith element in a void* array**
  ```c
  void *ith = (char *)base + i*elemsz;
  ```
The void* blues

- If typed pointers are dangerous, what about untyped ones…?
  - Pointer of any type is compatible

- What could possibly go wrong?
  - Mismatched pointee type
  - Size mismatched to pointee type
  - Wrong level of indirection on pointer
  - Mishandle manual scaling
  - Wrong typecast

- How do other languages support generic behavior?
To be wise in the ways of memory

- **Prefer array notation to pointer arithmetic where possible**
  
  arr[index]
  *(arr + index)
  *(type *)((char *)arr + index*elemSize)
  
  Same effect & efficiency, but subscript more readable, easier to get right

- **Use void * only when you must**
  
  If you know the type of pointee, don’t keep it a secret!

- **Drop down to memxxx functions only when you must**
  
  If you know the type being copied, use assignment (typecast if necessary)

- **Prefer stack to heap allocation where possible**
  
  Cheaper, more readable, less potential for error

- **Don’t store/declare/pass variables with extra levels of indirection**
  
  Use extra layer of pointer only when necessary

- **Use pointer typecasts exactly and only when required**
  
  Don’t ignore warnings about pointer mismatch, don’t cast indiscriminately
Go forth and * (dereference)

Your pointer questions here