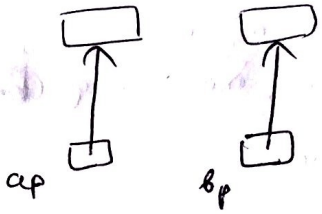
CS 107 lecture 4

lecture 4 generic algorithm implementation

non-generic version

```
void swap (int * ap, int * bp)
{
    int temp = *ap;
    *ap = *bp;
    *bp = temp;
}
```

implicit knowledge of how many bits are moved around



cannot swap doubles, structs, classes...

want a swap function that swaps to arbitrary figures of the same size

false generic version

```
void swap (void * vp1, void * vp2)
{   void temp = *vp1;
    *vp1 = *vp2;
    *vp2 = temp;
}
```

pointer of no specified type

false!!!

(!) cannot declare a variable of type void (!)

void:
1) void as just return type for fuctions stating that there is nothing to be returned.
2) argument to a function, saying that we do not expect anything
3) void * : generic pointer

(!) cannot dereference a void * pointer, as there is not information about # of bytes (!)

correct generic version

```
void swap (void * vp1, void * vp2, int size)
{
    char buffer[size];
    memcpy (buffer, vp1, size);
    memcpy (vp1, vp2, size);
    memcpy (vp2, buffer, size);
}
```

size of the figur

copy bytes, one byte a tim

← generic version of strcpy, not dedicated to characters, does not wait until \0

void and casting tells the compiler not to complain, but increases risks ↖ generics relax type checking

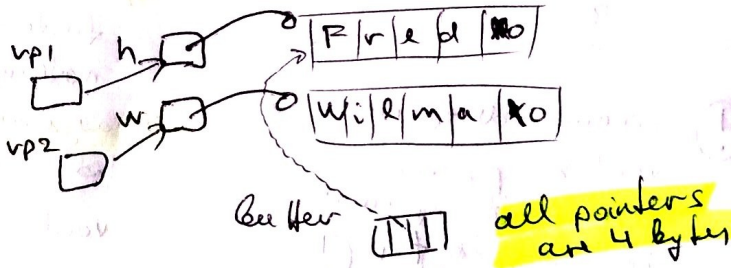
ex
```
int x=17, y=37;
swap (&x, &y, sizeof (int));
```

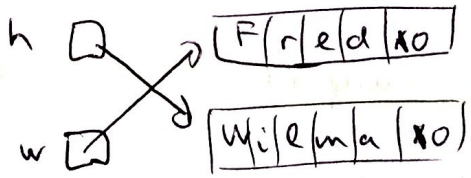
ex
```
double d=π, e=2;
swap (&d, &e, sizeof (double));
```

ex
```
int i=44;
short s=5;
swap (&i, &s, sizeof (short));
```
← will compile but is completely wrong

Swapping pointers

```
char * husband = strdup ("Fred");
char * wife    = strdup ("Wilma");
swap (&husband, &wife, sizeof (char *));
```
address of address





Wrong example:

```
swap (husband, wife, sizeof (char *))
```

non-generic

```
int lsearch (int key, int array[], int size)
{
    for (int i=0; i < size; i++) {
        if (array[i] == key) {
            return i;
        }
    }
    return -1;
}
```

== bitwise comparison

consequences of changing int to void: 1) cannot do pointer arithmetic of void* in array[i]
2) lose the ability to compare with == (e.g. not on strings)

generic version:
1) need to pass the size of each cell in the array
2) need to pass a comparison function

generic without function pointer

```
void* lsearch (void* key,
               void* base,
               int n,
               int elemSize)
{
    for (int i=0; i<n; i++) {
        void* elemAddr = (char*) base + i * elemSize;
        if (memcmp(key, elemAddr, elemSize) == 0) {
            return elemAddr;
        }
    }
    return NULL;
}
```

cannot do pointer arithmetic on void*.

# of bytes to move the pointer based on knowledge of size of elements

assigning char* to void* is fine; feel to more specific

memcmp: like string comparison, but not characters. compares elemSize bytes at key and elemAddr addresses. 0 if match, else 1 or -1

works for int, short, double, float ... not structs

[redacted] void * lsearch (void * key,
void * base,
int n,
int elemSize,
int (* cmpfn) [redacted] (void *, void *))

generic
with fxn pointer