

# Function pointers & void pointers

CS 261 Lab #6

```
#define TYPE int
```

```
struct Node{  
    struct Node *left;  
    struct Node *right;  
    TYPE value;  
}
```

← Set when we **compile**

```
int _binarySearch(TYPE *data, int size,  
                 TYPE val);
```

```
struct Node *_addNode(struct Node *cur, TYPE val);
```

Now we can't use `_binarySearch()` or  
`_addNode` in a BST for strings  
(this makes us sad)

What if we could tell those functions the type of data it will use at **runtime**?

We'd only have to write one version of the function, but we need to tell it how to **compare values** (numbers? strings?)

While we're at it, let's also tell it how **print values**... no more messing with %d/%s/%f each time we change TYPE!

**Function pointers** let us pass functions as parameters to other functions

*(they are pointers to functions)*

**Void pointers** let us use the same data type to store any type of data

*(they are pointers to unknown data)*

Together they let us build data types that can hold **any type** of data *(ints, strings, structs, etc.)*

~~#define TYPE int~~

TYPE data[];      ← Change to void pointer

int \_binarySearch(      ← Add function pointer  
    TYPE \*data, int size, TYPE val);

Array of void pointers

void\*\* data;

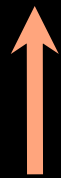
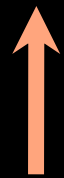
void pointer to search for

int \_binarySearch(  
    void \*\*data, int size, void \*val,  
    int(\*compareFunc)(void \*, void \*));

↑  
Pointer to a function that compares two values

The **type signature** of a function is the combination of parameters and return type

```
int compare(void *val1, void *val2);
```



This function's type signature has two void pointers  
and returns an int

The name (*compare*) doesn't matter; any  
function with the **same type signature**  
can be used with this function pointer

These are **void pointers** to match the definition of our **function pointer** (two void pointer parameters, returns an int)

```
int compareInts(void *val1, void *val2) {  
    int *intPtr1, *intPtr2;  
    int difference;  
  
    intPtr1 = (int *)val1;  
    intPtr2 = (int *)val2;  
  
    difference = *intPtr1 - *intPtr2;  
    return difference;  
}
```

Tell the compiler  
that these are  
actually **int  
pointers**

Now we can work with our int pointers  
like normal!

## Download code from CANVAS

See how we use void pointers and function pointers to use different types of data with the same **binarySearch()** function

Implement **compare** and **print** functions for **doubles** and **strings**  
(ints are already done as an example)

**Compare the runtimes** of **binarySearch()**  
and **sequentialSearch()**