Function Pointers

Two components

- Data

- Behavior/Functions

Linked List with int

```
typedef struct ListElement* ListElementPtr;
typedef struct ListElement
{
  int data;
  ListElementPtr *psNext;
} ListElement;

typedef struct List
{
  ListElementPtr psHead;
} List;
```

```
void printList(List *psList)
{
  ListElement *psTemp = psList->psHead;

  while( psTemp )
  {
    printInt(psTemp->data);
    psTemp = psTemp->psNext;
  }
}
```

Linked List with void*

```
typedef struct ListElement* ListElementPtr;
typedef struct ListElement
{
  void *pData;
  ListElementPtr *psNext;
} ListElement;

typedef struct List
{
  ListElementPtr psHead;
} List;
```

```
void walkList(List *psList)
{
  ListElement *psTemp = psList->psHead;

  while( psTemp )
  {
    printInt(psTemp->data);


    psTemp = psTemp->psNext;
  }
}
```

Function Pointers!

- Each function is stored at an address.
- Pointers are just an address
- We can have pointers to functions!

- Function pointers can be passed as arguments to other functions or return from functions

- Declare the function pointer variable

```
returnType (*variableName)(paramType ...);
```

Linked List with void*

```
void walkList(List *psList,
{
  ListElementPtr psTemp = psList->psHead;

  while( psTemp )
  {
    // printInt(psTemp->psData); ????


    psTemp = psTemp->psNext;
  }
}
```

print functions

```
// Shhhh! It's a
// secret to the List!
typedef struct Person
{
  //key
  char szName[MAX];

  //value
  int office;
} Person;
```

Design

- Separate data from structure
  - one function walks the linked list
    - always operates the same way
  - another function is used to interact with the data in each element

Other functionality

find

```
bool findInList(List *psList, void *pKey,
{
  ListElementPtr psTemp = psList->psHead;

  while( psTemp )
  {
    // did we find the node?
    if (
     {
       return true;
     }
     psTemp = psTemp->psNext;
  }
  return false;
}
```

Put it all together

```
typedef bool(*cmpEqual)(void*, void *);

typedef void (*userFunc)(void*);

typedef struct List
{
  cmpKey pKeyEqual;
  cmpKey pKeyGreater;

  userFunc pPrinter;

  ListElementPtr psHead;
} List;
```

```
bool lstFind(List *psList, void *pKey)
{
  ListElementPtr psTemp = psList->psHead;

  while( psTemp )
  {
    // did we find the node?
    if (
    {
      return true;
    }
    psTemp = psTemp->psNext;
  }
  return false;
}
```

Example: Make this generic

What happens if we replace **`int aInts[]`** with **`void* pArray`** ?

```
bool isSorted(int aInts[], int length)
{
  bool bSorted = true;

  int indx;

  for (indx = 0; indx < length - 1; ++indx)
  {
    if( aInts[indx] > aInts[indx + 1] )
    {
       bSorted = false;
    }
  }
 return bSorted;
}
```

Void * array

- How do we access elements in an array pointed to by a void*?

  int aInts[10];

  void* pArray = aInts;

  // print aInts[2]

  printf(“%d”,

Generic

```
bool genericIsSorted(void* pArray, int elementSize, int length,
                                                    // greaterThan
)
{
  bool bSorted = true;
    int indx, temp;

  for (indx = 0; indx < length - 1; ++indx)
  {
    if(



       )
    {

       bExchange = false;
    }
  }
  return bSorted;
}
```