Generics

Generic Swap

```
void swap (void *ap, void *bp, int size) {
    char temp[size];
    memcpy (temp, ap, size);
    memcpy (ap, bp, size);
    memcpy (bp, temp, size);
}

int main ()
{
    int i = 15;
    int j = 61;
    swap1 (&i, &j, sizeof (int));

    double x = 3.14156;
    double y = 61.2534;
    swap1 (&x, &y, sizeof (double));

    int p = 44;
    short q = 1;
    swap1 (&p, &q, sizeof (short));

    char *s1 = strdup ("Fred");
    char *s2 = strdup ("Wilma");
    swap1 (&s1, &s2, sizeof (char *));
    swap1 (s1, s2, sizeof (char *));
    swap1 (s1, &s2, sizeof (char *));
}
```

Linear Search

```
int linSearch (int key, int array[], int sizeOfArray) {
    int i;

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

int main () {
    int array[5] = {1, 2, -3, 44, -5};
    int key = 1;

    int status = linSearch (key, array, 5);

    if (status == -1)
        printf ("Not found");
    else
        printf ("%d\n", status);
}
```

Generic Linear Search

```
int genLinSearch (void *key, void *base, int sizeOfArray, int elemSize)
{
    int i;
    for (int i = 0; i < sizeOfArray; i++) {
       void *elemAddr = (char *) base + i * elemSize;
       if (memcmp (elemAddr, key, elemSize) == 0)
           return i;
    }
    return -1;
}

int main () {
    int array[5] = {1, 2, -3, 44, -5};
    int key = 1;

    int status = genLinSearch (&key, array, 5, sizeof (int));

    if (status == -1)
        printf ("Not found");
    else
        printf ("%d\n", status);
}
```

Function Pointers: Recap

int *gi();	() binds more tightly than *. *gi() is same as*(gi()) *(gi()) is of type int => gi() is a pointer to int, => gi is a function that returns pointer to int.
int (*hi)();	(*hi)() is int => (*hi) is a function that returns int => hi is a pointer to a function that returns int

Understanding typecasts

If you know how to declare a type, it is easy to write a cast for that type.

Type	Declaration	Cast
Pointer to function that returns int	int (*fp)()	int (*)()
Pointer to function that returns nothing	void (*fp)()	void (*)()

Function pointers

- Variables which point to the address of a function
- We can implement call-backs
- Complicated syntax
- Use it only when you need it!

Function pointer: Example

```
char ** fun()
{
   static char * z = (char*)"Merry Christmas :)";
   return &z;
}
int main()
{
   char ** ptr = NULL;

   char ** (*fun_ptr)(); //declaration of pointer to the function
   fun_ptr = &fun;

   ptr = fun();

   printf("\n %s \n Address of function = [%p]", *ptr, fun_ptr);
   printf("\n Address of first variable created in fun() = [%p]", (void*)ptr);
   cout<<endl;
   return 0;
}
```

More Generic Linear Search

```
int genLinSearch (void *key,void *base,int sizeOfArray, int elemSize, int (*cmpFunc) (void *, void *)) ) {
    int i;
    for (int i = 0; i < sizeOfArray; i++) {
       void *elemAddr = (char *) base + i * elemSize;
       if (cmpFunc(elemAddr, key) == 0)
           return i;
    }
    return -1;
}

int integerCmp (void *elem1, void *elem2) {
           int *ip1 = elem1;
           int *ip2 = elem2;

           return *ip1 - *ip2;
}

int main () {
    int array[5] = {1, 2, -3, 44, -5};
    int key = 1;

    int status = genLinSearch (&key, array, 5, sizeof (int), integerCmp);

    if (status == -1)
        printf ("Not found");
    else
        printf ("%d\n", status);
}
```

Fill out the body of stringCmp

```
int genLinSearch (void *key,void *base,int sizeOfArray, int elemSize, int (*cmpFunc) (void *, void *)) ) {
    int i;
    for (int i = 0; i < sizeOfArray; i++) {
       void *elemAddr = (char *) base + i * elemSize;
       if (cmpFunc(elemAddr, key) == 0)
           return i;
    }
    return -1;
}

int stringCmp (void *elem1, void *elem2) {
    char *s1 = * (char **) elem1;
    char *s2 = * (char **) elem2;

    return strcmp (s1, s2);

}

int main () {
    char *array[5] = {“Ab”, “F#”, “B”, “Gb”, “D”};
    char *key = “Eb”;

    int status = genLinSearch (&key, array, 5, sizeof (char *), stringCmp);

    if (status == -1)
        printf ("Not found");
    else
        printf ("%d\n", status);
}
```

With implicit casts

```
int genLinSearch (void *key,void *base,int sizeOfArray, int elemSize, int (*cmpFunc) (void *, void *)) ) {
    int i;
    for (int i = 0; i < sizeOfArray; i++) {
       void *elemAddr = (char *) base + i * elemSize;
       if (cmpFunc(elemAddr, key) == 0)
           return i;
    }
    return -1;
}

int stringCmp (void *elem1, void *elem2) {
    char **s1 = elem1;
    char **s2 = elem2;

    return strcmp (*s1, *s2);

}

int main () {
    char *array[5] = {“Ab”, “F#”, “B”, “Gb”, “D”};
    char *key = “Eb”;

    int status = genLinSearch (&key, array, 5, sizeof (char *), stringCmp);

    if (status == -1)
        printf ("Not found");
    else
        printf ("%d\n", status);
}
```

Is this fine?

```
int stringCmp (void *elem1, void *elem2) {
    char *s1 = elem1;
    char *s2 = elem2;

    return strcmp (s1, s2);

}

int main () {
    char *array[5] = {“Ab”, “F#”, “B”, “Gb”, “D”};
    char *key = “Eb”;

    int status = genLinSearch (&key, array, 5, sizeof (char *), stringCmp);

    if (status == -1)
        printf ("Not found");
    else
        printf ("%d\n", status);
}
```

Check this one !!

```
int stringCmp (void *elem1, void *elem2) {
    char *s1 = elem1;
    char *s2 = elem2;

    return strcmp (s1, s2);

}

int main () {
    char **notes = (char **) malloc (5 * sizeof (char *)); // Allocating 5 char pointers
    notes[0] = strdup ("Ab");
    notes[1] = strdup ("F#");
    notes[2] = strdup ("B");
    notes[3] = strdup ("Eb");
    notes[4] = strdup ("D");
    char *favNote2 = strdup ("Eb");
    status = genLinSearch (&favNote2, notes, 5, sizeof (char *), stringCmp);


    int status = genLinSearch (&key, array, 5, sizeof (char *), stringCmp);

    if (status == -1)
        printf ("Not found");
    else
        printf ("%d\n", status);
}
```