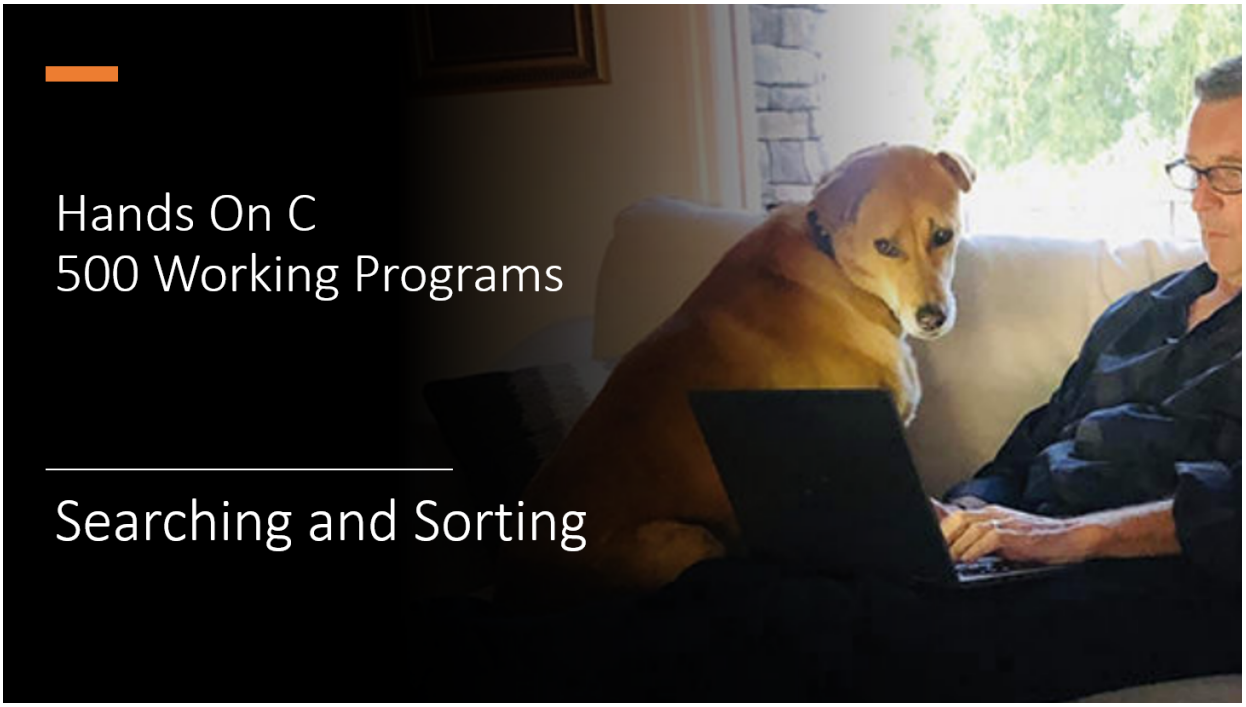


Searching an Array for a Value

In [1]:
```c
#include <stdio.h>

#define ARRAY_SIZE 5

int main(void)
{
  int values[ARRAY_SIZE] = { 30, 10, 40, 20, 50 };

  for (int i = 0; i < ARRAY_SIZE; ++i)
    if (values[i] == 20)
      printf("Found 20 at index %d\n", i);
}
```

```
Found 20 at index 3
```

In [2]:
```c
#include <stdio.h>
#include <string.h>

#define ARRAY_SIZE 5

int main(void)
{
  char *values[ARRAY_SIZE] = { "AAA", "CCC", "DDD", "BBB", "EEE" };

  for (int i = 0; i < ARRAY_SIZE; ++i)
    if (strcmp(values[i], "BBB") == 0)
      printf("Found BBB at index %d\n", i);
}
```

```
Found BBB at index 3
```

Using a Binary Search

In [3]:
```c
#include <stdio.h>

int binary_search(int array[], int value, int size)
 {
   int found = 0;
   int high = size, low = 0, mid;

   mid = (high + low) / 2;

   while ((! found) && (high >= low))
    {
      // printf("Low %d Mid %d High %d\n", low, mid, high);

      if (value == array[mid])
        found = 1;
      else if (value < array[mid])
        high = mid - 1;
      else
        low = mid + 1;
      mid = (high + low) / 2;
   }
  return((found) ? mid: -1);
 }

int main(void)
 {
   int values[100];

   for (int i = 0; i < 100; i++)
     values[i] = i;

   printf("Result of search for %d is index %d\n", 27, binary_search(values, 27,
   printf("Result of search for %d is index  %d\n", 75, binary_search(values, 75,
   printf("Result of search for %d is index  %d\n", 3, binary_search(values, 3, 1
   printf("Result of search for %d is index  %d\n", 500, binary_search(values, 50
 }
```

```
Result of search for 27 is index 27
Result of search for 75 is index  75
Result of search for 3 is index  3
Result of search for 500 is index  -1
```

Using the C Library lsearch Function

In [4]:
```
#include <stdio.h>
#include <search.h>

int compare(int *a, int *b)
{
   return (*a - *b);  // 0 if equal
}

void main ()
{
   int values[5] = {44, 69, 3, 17, 23};

   size_t elements = 5;

   int value_to_find = 17;
   int *value_location;

   value_location = (int *) lfind (&value_to_find, values, &elements, sizeof(int)

   if (value_location)
      printf (" %d found in linear search at offset %ld\n", value_to_find, value_
   else
      printf ("%d not found in linear search\n", value_to_find);
}
```

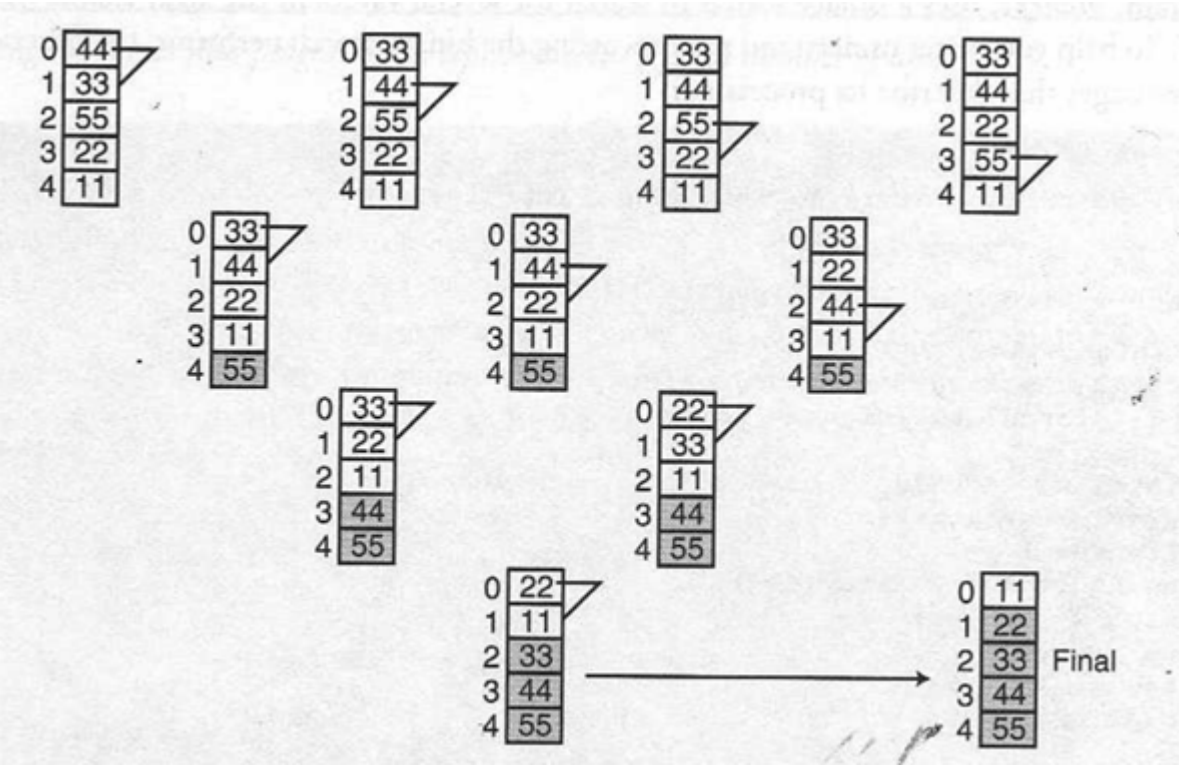
```
 17 found in linear search at offset 3
```

Using the C Libary bsearch Function

In [5]:
```
#include <stdio.h>
#include <stdlib.h>

int compare(const void * a, const void * b)
{
   return (*(int*) a - *(int*) b);   // 0 if same, negative if less, positive if 

}


int main (void)
{
   int values[] = { 51, 60, 78, 79, 93 };

   int *value_location;
   int value_to_find = 78;

   value_location = (int*) bsearch (&value_to_find, values, 5, sizeof(int), compa

  if (value_location != NULL )
      printf("Found item = %d at offset %ld\n", value_to_find, value_location - v
  else
      printf("Item = %d could not be found\n", value_to_find);
}
```

```
Found item = 78 at offset 2
```

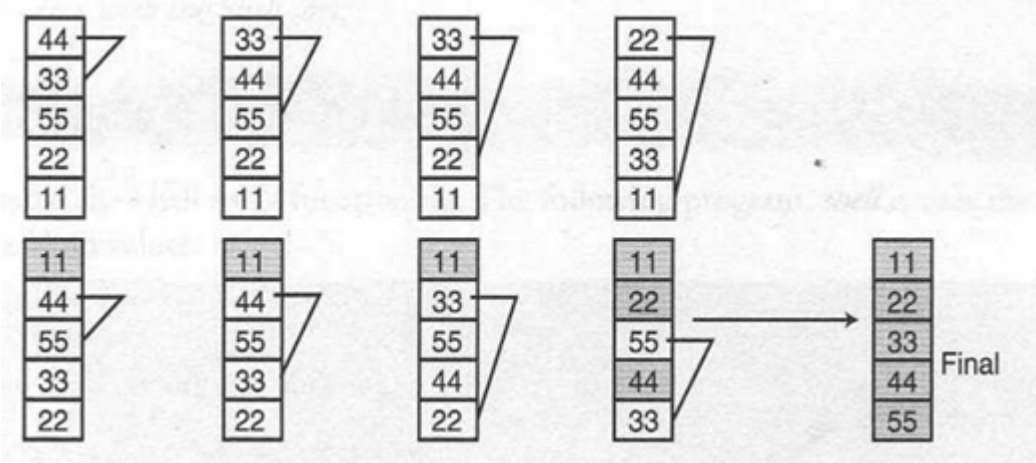
Understanding the Bubble Sort


0 44
1 33
2 55
3 22
4 11
0 33
1 44
2 55
3 22
4 11
0 33
1 44
2 55
3 22
4 11
0 33
1 44
2 22
3 55
4 11
0 33
1 44
2 22
3 11
4 55
0 33
1 44
2 22
3 11
4 55
0 33
1 22
2 44
3 11
4 55
0 33
1 22
2 11
3 44
4 55
0 22
1 33
2 11
3 44
4 55
0 22
1 11
2 33
3 44
4 55
0 11
1 22
2 33
3 44
4 55
Final

In [6]:
```c
#include <stdio.h>
#include <stdlib.h>

void bubble_sort(int array[], int size)
 {
   int temp, i, j;

   for (i = 0; i < size; i++)
    for (j = 0; j < size; j++)
      if (array[i] < array[j])
        {
          temp = array[i];
          array[i] = array[j];
          array[j] = temp;
        }
  }

int main(void)
 {
   int values[30], i;

   for (i = 0; i < 30; i++)
     values[i] = rand() % 100;

   bubble_sort(values, 30);

   for (i = 0; i < 30; i++)
     printf("%d ", values[i]);
 }
```

```
11 15 21 23 26 26 27 29 30 35 35 36 40 49 59 62 62 63 67 67 68 72 77 82 83 86 8
6 90 92 93
```

Understanding the Selection Sort



In [7]:
```
#include <stdio.h>
#include <stdlib.h>

void selection_sort(int array[], int size)
 {
   int temp, current, j;

   for (current = 0; current < size; current++)
    for (j = current + 1; j < size; j++)
      if (array[current] > array[j])
        {
          temp = array[current];
          array[current] = array[j];
          array[j] = temp;
        }
  }

int main(void)
 {
   int values[30], i;

   for (i = 0; i < 30; i++)
     values[i] = rand() % 100;

   selection_sort(values, 30);

   for (i = 0; i < 30; i++)
     printf("%d ", values[i]);
 }
```

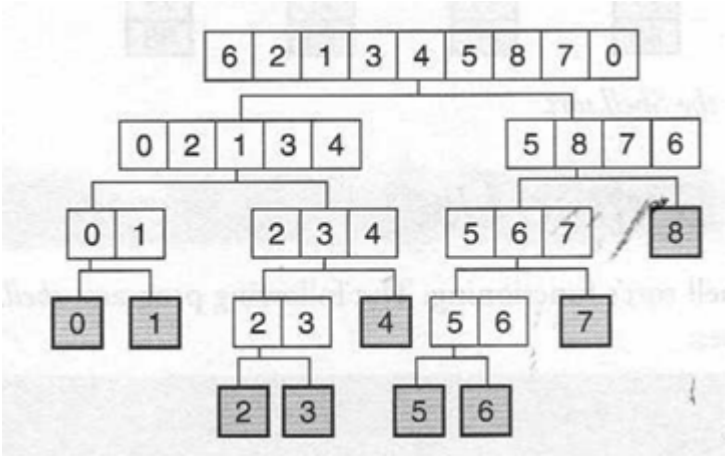
```
11 15 21 23 26 26 27 29 30 35 35 36 40 49 59 62 62 63 67 67 68 72 77 82 83 86 8
6 90 92 93
```

Understanding the Shell Sort

In [8]:
```
#include <stdio.h>
#include <stdlib.h>

void shell_sort(int array[], int size)
 {
   int temp, gap, i, exchange_occurred;

   gap = size / 2;

   do
   {
    do
     {
      exchange_occurred = 0;

      for (i = 0; i < size - gap; i++)
       if (array[i] > array[i + gap])
        {
          temp = array[i];
          array[i] = array[i + gap];
          array[i + gap] = temp;
          exchange_occurred = 1;
        }
    } while (exchange_occurred);
   }
  while (gap = gap / 2);
  }

int main(void)
 {
   int values[50], i;

   for (i = 0; i < 50; i++)
     values[i] = rand() % 100;

   shell_sort(values, 50);

   for (i = 0; i < 50; i++)
     printf("%d ", values[i]);
 }
```

```
2 11 11 15 15 19 21 21 22 23 24 26 26 27 29 29 29 30 35 35 36 37 40 42 49 56 58
59 62 62 63 67 67 67 68 69 70 72 73 77 82 83 84 86 86 90 92 93 93 98
```

Understanding the Quick Sort



In [9]:
```
#include <stdio.h>
#include <stdlib.h>

void quick_sort(int array[], int first, int last)
 {
   int temp, low, high, list_separator;

   low = first;
   high = last;
   list_separator = array[(first + last) / 2];

   do
   {
     while (array[low] < list_separator)
       low++;

     while (array[high] > list_separator)
       high--;

     if (low <= high)
      {
        temp = array[low];
        array[low++] = array[high];
        array[high--] = temp;
      }
   }
   while (low <= high);

  if (first < high)
    quick_sort(array, first, high);
  if (low < last)
    quick_sort(array, low, last);
 }

void main(void)
 {
   int values[100], i;

   for (i = 0; i < 100; i++)
     values[i] = rand() % 100;

   quick_sort(values, 0, 99);

   for (i = 0; i < 100; i++)
     printf("%d ", values[i]);
 }
```

```
2 3 5 5 8 11 11 12 13 13 14 15 15 19 21 21 22 23 24 24 25 26 26 26 26 27 27 29
29 29 29 30 32 34 35 35 36 36 37 39 39 40 42 43 45 46 49 50 51 54 56 56 57 58 5
9 60 62 62 62 63 64 67 67 67 67 68 68 69 70 70 72 73 73 76 76 77 78 80 81 82 82
83 84 84 84 86 86 86 87 88 90 91 92 93 93 94 95 96 98 99
```

Sorting an Array of Character Strings

In [10]:
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void bubble_sort(char *array[], int size)
 {
   char *temp;
   int i, j;

   for (i = 0; i < size; i++)
    for (j = 0; j < size; j++)
      if (strcmp(array[i], array[j]) < 0)
        {
          temp = array[i];
          array[i] = array[j];
          array[j] = temp;
        }
  }

int main(void)
 {
   char *values[] = {"AAA", "CCC", "BBB", "EEE", "DDD"};
   int i;

   bubble_sort(values, 5);

   for (i = 0; i < 5; i++)
     printf("%s ", values[i]);
 }
```

```
AAA BBB CCC DDD EEE
```

Using the C Library qsort

```
Compare Function Returns

<0 The element pointed by a goes before the element pointed by b
0  The element pointed by a is equivalent to the element pointed by b
>0 The element pointed by a goes after the element pointed by b
```

In [11]:
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int int_compare (const void *a, const void *b)
{
   return (*(int *)a - *(int *)b);
}

int float_compare (const void *a, const void *b)
{
  if (*(float *) a < *(float *) b)
    return(-1);
  else if (*(float *) a > *(float *) b)
    return(1);
  else
    return(0);
}

int string_compare (const void *a, const void *b)
{
    const char* string1 = *(const char**)a;
    const char* string2 = *(const char**)b;
    return strcmp(string1, string2);
}

#define ARRAY_SIZE 5

int main (void)
{
   int values[] = { 99, 33, 80, 12, 5 };
   float float_values[] = { 1.2, 3.1, 2.2, 5.5, 4.4 };
   char *strings[] = { "AAA", "DDD", "BBB", "EEE", "CCC" };

   qsort(values, ARRAY_SIZE, sizeof(int), int_compare);
   qsort(float_values, ARRAY_SIZE, sizeof(float), float_compare);
   qsort(strings, ARRAY_SIZE, sizeof(char *), string_compare);

   for(int i = 0; i < 5; i++ )
      printf("%d ", values[i]);

   printf("\n");

   for(int i = 0; i < 5; i++ )
      printf("%5.2f ", float_values[i]);

   printf("\n");

   for(int i = 0; i < 5; i++ )
      printf("%s ", strings[i]);
}
```

```
5 12 33 80 99
 1.20  2.20  3.10  4.40  5.50
AAA BBB CCC DDD EEE
```

Reversing the Sort Order

In [12]:
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int int_compare (const void *a, const void *b)
{
   return (*(int *)b - *(int *)a);
}

int float_compare (const void *a, const void *b)
{
  if (*(float *) b < *(float *) a)
    return(-1);
  else if (*(float *) b > *(float *) a)
    return(1);
  else
    return(0);
}

int string_compare (const void *a, const void *b)
{
    const char* string1 = *(const char**)a;
    const char* string2 = *(const char**)b;
    return strcmp(string2, string1);
}

#define ARRAY_SIZE 5

int main (void)
{
   int values[] = { 99, 33, 80, 12, 5 };
   float float_values[] = { 1.2, 3.1, 2.2, 5.5, 4.4 };
   char *strings[] = { "AAA", "DDD", "BBB", "EEE", "CCC" };

   qsort(values, ARRAY_SIZE, sizeof(int), int_compare);
   qsort(float_values, ARRAY_SIZE, sizeof(float), float_compare);
   qsort(strings, ARRAY_SIZE, sizeof(char *), string_compare);

   for(int i = 0; i < 5; i++ )
      printf("%d ", values[i]);

   printf("\n");

   for(int i = 0; i < 5; i++ )
      printf("%5.2f ", float_values[i]);

   printf("\n");

   for(int i = 0; i < 5; i++ )
      printf("%s ", strings[i]);
}
```

```
99 80 33 12 5
 5.50  4.40  3.10  2.20  1.20
EEE DDD CCC BBB AAA
```

What You will Learn Next

Arrays let your programs store multiple values of the same type. A problem with using arrays is that your programs must know in advance the number of elements the array must store. In the next lesson, you will learn how to allocate memory dynamically as the program executes and how to use that memory to create linked lists and binary trees.