August 22, 2020

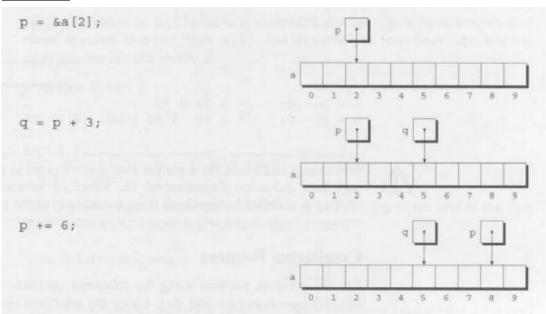
- 1. a) 14
 - b) 34
 - c) 4
 - d) true
 - e) false

Notes

• Pointer Arithematic

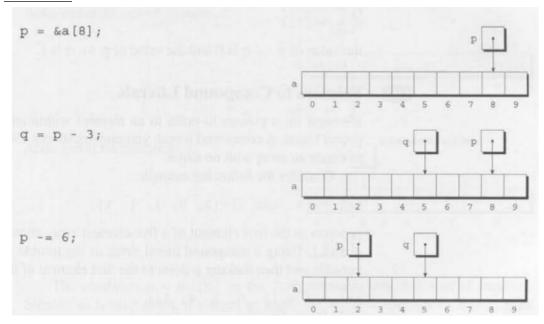
- Adding an integer to a pointer

Example



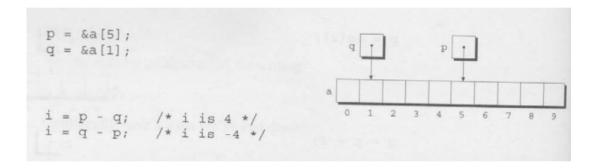
- Subtracting an integer from a pointer

Example



- Subtracting one pointer from another

Example



• Comparing pointers

- Can compare pointers using relational operators (i.e. <,<=,>,>=) and the equality operators (i.e. ==,!=)
- Returns 1 if true and 0 if false

Example

```
p = &a[5];

q = &a[1];

p <= q \text{ is } 0 \text{ and } p >= q \text{ is } 1
```

2. low and high are memory addresses.

So, low + high is out of bound, and it could potentially point to an undesirable or wrong value.

To fix this, we subtract the from high value to the low value:

$$\mathtt{middle} = \frac{\mathtt{low} \; + \; \mathtt{high}}{2} \tag{1}$$

3. I need to write the contents of an array a after the execution of statements outlined in problem sheet.

After execution, the array would have contents of [10, 9, 8, 7, 6, 5, 4, 3, 2, 1].

Notes

- Combining the * and ++ Operators
 - *p++ or *p++ \rightarrow Value of expression is *p before increment; increment p later
 - (*p)++ \rightarrow Value of expression is *p before increment; increment *p later
 - -*++p or $*(++p) \rightarrow$ Increment p first; value of expression is *p after increment
 - $++*p \text{ or } ++(*p) \rightarrow \text{Increment } *p \text{ first; value of expression is } *p \text{ after increment}$

Example

$$a[i++] = j$$

Means assign the value j to a[i] before increment

Example 2

```
for (p = &a[0]; p < &a[N]; p++)
sum += *p;
```

Is the same as

```
p = &a[0];
while (p < &a[N])
  sum += *p++;</pre>
```

4. I need to re-write prototype make_empty, is_empty and is_full of the following code to use the pointer variable top_ptr instead of the integer variable top.

```
#include <stdbool.h>
1
2
      #define STACK_SIZE 100
3
      /*external variables*/
5
      int contents[STACK_SIZE]
      int top = 0;
8
      void make_empty(void) {
9
           top = 0;
10
      }
11
12
      bool is_empty(void) {
13
          return top == 0;
14
15
16
      bool is_full(void) {
17
          return top == STACK_SIZE;
18
19
```

And after re-write using top_ptr instead of top have:

```
#include <stdbool.h>
2
      #define STACK_SIZE 100
3
4
      /*external variables*/
5
      int contents[STACK_SIZE]
6
      int *top_ptr = &contents[0];
8
      void make_empty(void) {
9
          top_ptr = &contents[0];
11
12
      bool is_empty(void) {
13
          return top_ptr == &contents[0];
14
15
16
      bool is_full(void) {
17
          return top_ptr == &contents[STACK_SIZE-1];
19
```

5. First, I need to identify which of the following expressions are illegal because of mismatched types.

```
a) p == a[0]
```

b)
$$p == &a[0]$$

```
c) *p == a[0]
```

$$d) p[0] == a[0]$$

Here, only a) is illegal.

Second, I need to write which of the remaining expressions are true.

Here, the expressions that return true are b), c) and d).

Notes

- *(a+i) is equal to a[i]
- *p and a[] are the same given p == a
- Using an Array Name as a Pointer
 - The name of an array can be used as a pointer to the first element in the array.

Example

```
int a[10];
*a = 7; /* stores 7 in a[0] */
*(a+1) = 12; /* stores 7 in a[1] */
```

Example 2

```
To simplify the loop, we can replace &a [0] by a and &a [N] by a + N: for (p = a; p < a + N; p++) sum += *p;
```

6. I need to re-write the following to use pointer arithematic instead of array subscripting, and I need to make as few change as possible.

```
int sum_array(cost int a[], int n) {
    int i, sum;

sum = 0;

for (i = 0; i < n; i++)
    sum += a[i];

return sum;
}</pre>
```

After making changes to above code to use pointer arithematic, we have

```
int sum_array(cost int a[], int n) {
   int i, sum;

sum = 0;

for (i = 0; i < n; i++)
   sum += *(a+i);

return sum;

}</pre>
```

7. I need to write the following using pointer arithematic so it finds an element in a that matches to value key. I need to return true if there is a match.

bool search(const int a[], int n, int key);

And the solution is:

```
bool search(cost int a[], int n, int key) {

for (int i = 0; i < n; i++) {
    if (*(a+i) == key) {
        return true
    }
}

return false;
}</pre>
```

8. Here, I need to re-write the following function to use pointer arithmetic instead of array subscripting.

```
void store_zeros(cost int a[], int n) {
   int i;

for (i = 0; i < n; i++) {
      a[i] = 0;
   }
}</pre>
```

After re-writing above code, we have

```
void store_zeros(cost int a[], int n) {
    int *p;

for (p = a; p < a + n; p++) {
        *p = 0;
    }
}</pre>
```

9. Here, I need to write the function

```
double inner_product(const double *a, const double *b, int n) using pointer arithmetic such that it returns a[0] * b[0] + a[1] * b[1] + a[2] * b[2] + ... + a[n-1] * b[n-1].
```

The solution is provided below

```
double inner_product(const double *a, const double *b, int n) {
           double sum = 0, p*;
2
3
          p = a;
5
           while (p < a + n) {
6
               sum += *a * *b;
               a++;
9
               b++;
10
11
               p++;
           }
12
13
           return sum;
```

```
Correct Solution

double inner_product(const double *a, const double *b, int n) {
    double sum = 0, *p;

    p = a;

    while (p++ < a + n) {
        sum += *a++ * *b++;
    }

    return sum;
}</pre>
```

10. Here, I need to rewrite the function find_middle so that it uses pointer arithmetic - not subscripting - to visit any element.

The solution to this exercise is provided below.

```
int *find_middle(int a[], int n) {
    return a + (n/2)
}
```

11. Here I need to modify the function find_largest function so that it uses pointer arithmetic - not subscripting - to visit array elements

```
int *find_largest(int a[], int n) {
           int *p, *max;
2
3
           max = a;
4
5
6
           for (p = a; p < a + n; p++){
               if (*p > *max) {
                    max = p;
               }
9
           }
10
11
           return *max;
12
```

12. I need to write the function

void find_two_largest(const int *a, int n, int *largest, int *second_largest)
using pointer arithmetic.

The solution to this exercise is:

```
#include <stdbool.h> // bool
      #include <limits.h> // INT_MIN
2
3
      bool is_largest(int current_max, int val);
4
5
      void find_two_largest (int a[], int n, int *largest, int*
6
     second_largest) {
          int current_max = INT_MIN;
          int current_second_max = INT_MIN;
8
9
          for (int i = 0; i < n; i++) {</pre>
               if (is_largest(current_max, a[i])) {
                   current_second_max = current_max;
                   current_max = *(a + i);
13
               }
14
          }
15
17
          *largest = current_max;
          *second_largest = current_second_max;
18
      }
19
20
      bool is_largest(int current_max, int val) {
21
          if (val > current_max) {
22
               return true;
23
          }
24
25
```

```
26     return false;
27 }
```

```
Correct Solution:
      void find_two_largest(const int *a, int n, int *largest, int *
     second_largest) {
          const int *p = a;
3
          *largest = *second_largest = *a;
          while (p++ < a + n) {
              if (*p > *largest) {
                  *second_largest = *largest;
                  *largest = *p;
              } else if (*p > *second_largest)
                  *second_largest = *p;
11
          }
12
      }
13
```

13. I need to rewrite the following program such that it uses a single pointer through the array one element at a time.

```
#define N 10
1
2
      double ident[N][N];
3
      int row, col;
4
      for (row = 0; row < N; row++) {
6
           for (col = 0; col < N; col++) {</pre>
               if (row == col) {
8
                    ident[row][col] = 1.0;
9
               } else {
10
                    ident[row][col] = 0.0;
11
               }
12
           }
13
```