

Here are the new push and pop functions (updating the other stack functions is left as an exercise):

```
void push(int i)
{
    if (is_full())
        stack_overflow();
    else
        *top_ptr++ = i;
}

int pop(void)
{
    if (is_empty())
        stack_underflow();
    else
        return *--top_ptr;
}
```

Note that I've written `*--top_ptr`, not `*top_ptr--`, since I want pop to decrement `top_ptr` *before* fetching the value to which it points.

12.3 Using an Array Name as a Pointer

Pointer arithmetic is one way in which arrays and pointers are related, but it's not the only connection between the two. Here's another key relationship: *The name of an array can be used as a pointer to the first element in the array.* This relationship simplifies pointer arithmetic and makes both arrays and pointers more versatile.

For example, suppose that `a` is declared as follows:

```
int a[10];
```

Using `a` as a pointer to the first element in the array, we can modify `a[0]`:

```
*a = 7;    /* stores 7 in a[0] */
```

We can modify `a[1]` through the pointer `a + 1`:

```
*(a+1) = 12;    /* stores 12 in a[1] */
```

In general, `a + i` is the same as `&a[i]` (both represent a pointer to element `i` of `a`) and `*(a+i)` is equivalent to `a[i]` (both represent element `i` itself). In other words, array subscripting can be viewed as a form of pointer arithmetic.

The fact that an array name can serve as a pointer makes it easier to write loops that step through an array. Consider the following loop from Section 12.2:

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