PH502: Scientific Programming Concepts

Irish Centre for High End Computing (ICHEC)

September 23, 2020

Overview

- In this lecture we are going to revisit pointers.
- We will recap on some of the basics.
- Pointers are not only used to pass information out of functions in C but are also used together with arrays.
- In C variable arithmetic means that pointers and arrays are in fact synonymous.

Pointers

- A pointer is a variable type that stores a memory address.
- This address can be used to read or modify the memory content in this particular location.
- The * and & operators allow us to manipulate pointers:
  - Used in a variable declaration, * marks a variable type as a pointer.
  - Used anywhere else, * is the *dereferencing* operator that accesses the value stored at the address stored by the pointer variable.
  - & is the pointer generating operator, used to generate a pointer to a variable's value.

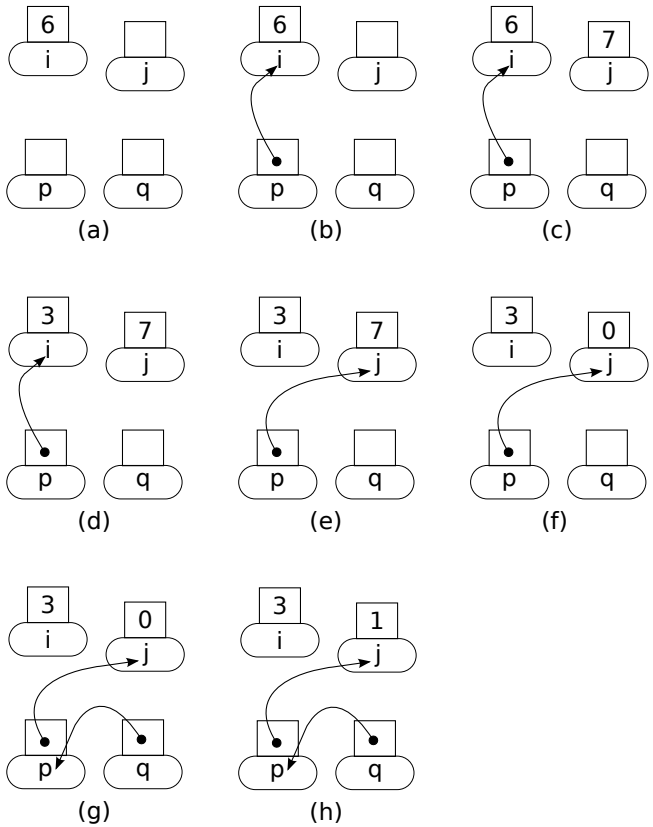
```
int i = 3;
int   *ip; // ip is a pointer to an integer
float *fp; // fp is a pointer to a float

ip = &i;   // ip points to the content of i
i == *ip;  // true
fp = &i;   // type error
```

Pointers (examples)

```
int *p, **q;
int i, j;

i    = 6;        // (a)
p    = &i;       // (b)
j    = *p + 1;   // (c)  j==7
*p   = 3;        // (d)  i==3
p    = &j;       // (e)
*p   = 0;        // (f)  j==0
q    = &p;       // (g)
**q = *p + 1;    // (h)  j==1
```



Pointers (examples)

- When declaring a pointer, whitespace between the star marker `*` and the variable name is ignored.
- That said, *it doesn't give a line-wide scope to the star marker*.
- Easy way to avoid any issues: favour use of the `type *var` syntax rather than `type* var`.

```
int *p;           // p is a pointer to an int
int* q;           // q is a pointer to an int
int* r, s;        // r is a pointer to an int
                  //     but s is just an int
int* *t, u, *v; // what's t? u? v?
```

Pointers and Arrays

- Each cell of a C array has its own address in memory.
- A pointer doesn't have to point at a variable, therefore can be used to reference a particular array cell.

```c
int arr[10];
int *p;
p = &arr[2];
```

```fortran
integer (kind=4), target :: arr(10)
integer (kind=4), pointer :: p
p => arr(3)
```

- The pointer `p` in the example above points to the third cell of the integer array `arr`.

Pointer Arithmetic

- A pointer being a numerical address in memory, it is possible to apply basic arithmetic on it.

```
int i, *p;

i = 42;
p = &i;
printf("p   points to address: %p.\n", p);
printf("p+1 points to address: %p.\n", p+1);

// sample output:
// p   points to address: 0xbfdf7028.
// p+1 points to address: 0xbfdf702c.
```

- What is the value of *p? *(p+1)?
- Looking at the example above, p+1 is actually an increment of 4 over the hexadecimal value of p.
- This is because pointer arithmetic is *type-dependent* and a pointer increment will see the address increase by as many bytes required to store the type of variable pointed at. Here, an integer is stored on 4 bytes.

Pointer Arithmetic

- C arrays are stored as a contiguous sequence of bytes in memory.
- Therefore, if one has access to one cell, pointer arithmetic allows you to access all other cells.

```
int a[10], i, *p, *q, *r;

p = &a[2]; // points to 3rd cell
q = p + 3; // points to 6th cell
r = p - 2; // points to 1st cell
i = q - r; // works too. value of i?
```

Pointer Arithmetic

- C offers some convenient equivalences between pointers and arrays.

```
int a[10], c, *p;

p = &a[0]; // point p to first cell
p = a;     // equivalent to previous line

c = a[3];   // c takes the value of the content
            // of the 3rd cell
c = *(p+3); // pointer arithmetic equivalent
c = p[3];   // quicker equivalent notation
```

- These equivalences are fundamental when passing pointers or arrays as function arguments.

Pointer Arithmetic (exercise)

- What are the values in a and what is p pointing at at the various points in the code?

```
int a[3], i, *p;

for (i=0; i<=2; i++) { a[i] = 0 } // (a)
p = &a[0];                        // (b)
p++;                              // (c)
*(p++);                           // (d)
*p++;                             // (e)
```

The NULL Pointer

- NULL is a special value for a pointer not pointing anywhere.
- Actual value in memory meaning 'NULL' is 0.

```
int *p = NULL;
int *q = 0; // same
```

```
integer (kind=4), pointer :: p
p => NULL()
```