Software Tools

C, Unix (Linux), and tools

Pointers

- Pointers are among the most powerfull aspects of C
- Require disciplined programming and careful testing
- C has many characteristics that make it different than other languages:
  - Pointers and arrays are very closely related
  - Pointer arithmetic
  - Extremely flexible declaration system

Memory model

- The memory (from a programmers point of view) is a long series of bytes each one numbered from zero to several trillions (for a 48 bit address space)
- The zeroth byte is not used by convention
  - This is the `NULL` pointer
- The rest can be used if the OS has given the OK
  - If we try to use a byte that has not been Okeyed we get “Segmentation Violation”
- A `char` is a single byte, a `short` two bytes, an `int` 4, a `float` 4, a `double` 8, etc.

The address of a variable

- An address looks like an integer, but is treated differently
- The adress of an `int` is the address of the “first” byteWe can get the address of a variable with the prefix operator `&`.
  - `c = 3;`
  - `ptr = &c;`
- We get the contents of an address with `*`.
  - `*ptr` is equal to 3.

How to read these symbols

- We read `&c` as the address of `c`.
- We read `*ptr` as the contents of address `ptr`.
- So if we read
  - `int k;`
- As `k` is an integer, then we read
  - `int *ptr;`
- As the content of address `ptr` is an integer.

Examples

- If we have the following definitions
  - `int x, *ptr;`
- Then the statements
  - `ptr = &x;`
  - `*ptr = 5;`
- Result in `x==5`. And
  - `*ptr +=3;`
- Result in x becoming 8.

Pointers and Function Arguments

- Lets try to write a swap function
- Does not work!

```
void badswap(int x, int y)
{
  int t;

  t = x;
  x = y;
  y = t;
}
```

Try again

- Here we declare pointers to the variables
- And it works!

```
void swap(int *xp, int *yp)
{
  int t;

  t = *xp;
  *xp = *yp;
  *yp = t;
}
```

```
{
  int x, y;

  x = 3;
  y = 2;
  swap(&x, &y);
}
```

Pointer and Array Magic

- Pointers and arrays are almost the same in C
- If we write
  - `int *p, a[10];`
  - `p=a;`
- Then
  - `*p == a[0];`
  - `*a == a[0]`
  - `*(a+1) == a[1];`
  - `&a[2] == a+2;`

Some Other Cryptic Stuff

- The definition
  - `char s[ ];`
- Is the same as
  - `char *s;`
- We can have negative indices!
  - `s[-1];`
- As long as we are sure the element exists.

Problems

- Write a recursive function rec_rev(char s[], int sz) the reverses a string s[] of sixe sz recursively in place.
- Write a recursive prefix expression calculator that evaluates expressions recursively. So something like + 2 * 3 4 would evaluate to 14