Australian National University

Pointers

Eric McCreath

Australian National University

Introduction

Pointers in *c* are basically memory addresses, they provide a simple and flexible way of your program referencing data.

All parameters in *c* are passed by making copies of the arguments to the function call. Hence pointers can be used to return information to a calling function.

Note that the pointer to *x* is denoted *&x* . And given a pointer *p* then the item it points to is denoted $*p$.

```
#include<stdio.h>
void addto(int *sum, int x) {
    *sum += x;
}
main() {
    int x, sum;
    sum = 0;
    while(scanf("%d",&x) == 1) {
        addto(&sum,x);
    }
    printf("Sum = %d",sum);
}
```

Australian National University

Its just memory

When working with pointers keep remembering data just sits in memory. Pointers give you a good way of referencing data within this memory.

```
    int val=0;
    int *pointer;
    pointer = &val;
    *pointer = 7;
    printf("%d",val); // will print 7
```

Australian National University

Arrays and pointers

Arrays are basically just pointers dressed up with a little more syntax.

```
   int array[] = {7,4,2,6};
   int *pointer;
   pointer = array; // or pointer = &array[0] is the same
   printf("%d",*pointer); // prints 7
   pointer += 2;
   printf("%d",*pointer); // prints 2
```

When possible stick with the array syntax for arrays as it is a little clearer for people reading your code.

Australian National University

NULL

NULL, which is just 0, is used as a marker to say empty or not set. Memory is generally not allocated to programs at address 0 so it can be used as a marker.

```
    int *pointer;
    pointer = NULL;
    ...
    if (!pointer) { // check that the pointer points to something
```

Pointers can point to pointers

Pointers can point to pointers. This can be useful if you have a number of parts of you program reference the same data.

```
   int value1 = 7;
   int value2 = 9;
   int *pointer;
   pointer = &value1;
   int **ppointer1;
   int **ppointer2;
   ppointer1 = &pointer;
   ppointer2 = &pointer;
   printf("%d",**pointer1); // prints 7
   printf("%d",**pointer2); // prints 7
   pointer = &value2;
   printf("%d",**pointer1); // prints 9
   printf("%d",**pointer2); // prints 9
```

Exercises

- The below code has an array of pointers that point to an array of integers. Add some code between the show methods such that the list will be shown in sorted order without moving the integers in the "array" (this involves modifying the "parray" such that it points to the elements of "array" in sorted order).

```
#include<stdio.h>
#define SIZE 4
void show(int *pa[], int len) {
    int i;
    for (i=0;i<len;i++) printf("%d ", *pa[i]);
    printf("\n");
}
int main() {
  int array[] = {4,7,3,8};
  int *parray[SIZE];
  int i;
  for (i=0;i<SIZE;i++) parray[i] = &array[i];
  show(parray,SIZE);
  // add your code here
  show(parray,SIZE);
  return 0;
}
```