CSC 4630

Supplementary Material on C

Review of Pointers
Checklist on pointers in C (do you remember how these work?)

* and & operators

NULL and undefined values

assigning values to pointers
    p = &v;
    p = malloc(sizeof(int));    /* (<stdlib.h>; (int *) not necessary) */

pointers and arrays - the same but different
    *(p+i) == p[i]

pointer arithmetic

strings and their implementation

pointers as function arguments
```c
$ cat pointers.c
/* pointers.c (not so intimidating) */

int n;
int *p;

int main(void)
{
    n = 4630;
p = &n;       /* address of n */
    printf("n is %d\n", n);
    printf("n's address is %ld\n", &n);
    printf("p holds n's address, so its value of course is %ld\n", p);
    printf("dereferencing p gives us %d\n", *p);
    printf("p also has an address, which is %ld\n", &p);

    return 0;
}

$ pointers
n is 4630
n's address is 137924
p holds n's address, so its value of course is 137924
dereferencing p gives us 4630
p also has an address, which is 137928
$
Pointers Overview

- What is a pointer?
  A variable that contains an address

- Why use pointers?
  To efficiently access data
  To write flexible code
  To change variables passed to a function
  To work with dynamically allocated memory
  To access hard-coded addresses in system code
/* two int's and a pointer to an int */
int a, b, *p;

a = b = 7;

/* p is assigned the address of a */
p = &a;

/* Object pointed to by p (i.e. a) gets 3 */
*p = 3;

<table>
<thead>
<tr>
<th>a:</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b:</td>
<td>7</td>
</tr>
<tr>
<td>p:</td>
<td></td>
</tr>
</tbody>
</table>
/* two int's and a pointer to an int */
int a, b, *p;

a = b = 7;

/* p is assigned the address of a */
p = &a;

/* Object pointed to by p (i.e. a) gets 3 */
*p = 3;

/* Now p is changed to point to b */
p = &b;

/* input integer into address of b */
scanf("%d", p);

\begin{tabular}{l}
  a: & 3 \\
  b: & 9 \\
  p: & \\
\end{tabular}
Declaring, Initializing, and Using Pointers

#include <stdio.h>

main()
{
    int num1=3, num2=6;
    int *p;

    p = &num1;
    printf("%d\n", *p);

    *p = 20;
    printf("%d ", *p);
    printf("%d\n", num1);

    p = &num2;
    printf("%d\n", *p);
}

Output:
3
20 20
6
The Null Pointer

- It is not legal to read from or write to address 0

- A null pointer:
  
  Pointer with value of 0
  Returned by pointer-returning functions to signify failure

- NULL is defined in stdio.h as

  ```c
  #define NULL     0
  -or-
  #define NULL    (void *) 0
  ```
### POINTERS AND ADDRESSES (Contd)

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>DECLARATION</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>int a;</td>
<td>3000410</td>
</tr>
<tr>
<td>b</td>
<td>int b;</td>
<td>3000414</td>
</tr>
<tr>
<td>c</td>
<td>int *c;</td>
<td>3000420</td>
</tr>
</tbody>
</table>

**INITIALIZE:**
- a = 1;
- b = 2;
- c = &a;

**INITIALLY:**
- &a is 3000410
- a is 1
- &b is 3000414
- b is 2
- &c is 3000420
- c is 3000410
- *c is 1

**CHANGE a = 3:**
- a is 3
- b is 2
- *c is 3

**CHANGE c = &b:**
- a is 3
- b is 2
- c is 3000414
- *c is 2
Pointer Arithmetic

- Important to declare pointer with correct type
- OK to add to, subtract from, compare pointers and subtract one pointer from another

```c
char *p;
*p is a character
p++ is equivalent to p = p + (1 byte)

int *p;
*p is an integer
p++ is equivalent to p = p + (bytes in an int)
```
## POINTER SCALE FACTOR

<table>
<thead>
<tr>
<th>POINTER TO</th>
<th>SCALE FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTER</td>
<td>1</td>
</tr>
<tr>
<td>SHORT</td>
<td>2</td>
</tr>
<tr>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>LONG</td>
<td>4</td>
</tr>
<tr>
<td>FLOAT</td>
<td>4</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>8</td>
</tr>
<tr>
<td>ARRAY</td>
<td>SIZE OF ARRAY ELEMENT</td>
</tr>
<tr>
<td>STRUCTURE</td>
<td>SIZE OF STRUCTURE (INCLUDING HOLES)</td>
</tr>
</tbody>
</table>
/* Declare an array and two pointers */
int A[10], *p0, *p1;

/* Set p0 to point to first element of A */
p0 = A;

/* Set p1 to point to second element of A */
p1 = A + 1;
POINTER VS ARRAYS

- Declaration

    char name[35];    /* storage for 35 bytes */
    char *p;         /* storage for 1 address */

- Assignment

    p = name;        /* Pointer initialization */
    /* required before */
    /* p can be used */

    name = address    /* Illegal */

- The [ ] and * operators may be used with both arrays and pointers.

  - Convention to use

    [ ] with arrays
    name[i] = 's';

    * with pointers
    p = &name[i];
    *p = 's';
Using a Pointer to Traverse an Array

- For sequential access, * is faster than []

```c
/* Reads an input line into an array */
/* Assumes input line is <= LINELEN chars */

#include <stdio.h>
define LINELEN 256

main()
{
    char line[LINELEN + 1], *p;
    p = line;    /* p = &line[0]; */
    while ((*p = getchar()) != '\n')
        p++;
    *p = '\0';
    printf("%s\n", line);
}
```

$ a.out
do it  <-- typed by user
do it  <-- output from program
$
/* swap happens in the body of main() */

#include <stdio.h>

int main(void)
{
    int i = 10, j = 20, temp;

    printf("In the beginning, i = %2d, j = %2d\n", i, j);

    temp = i;     /* Here is the "swap" */
i = j;
j = temp;

    printf("At the end, i = %2d, j = %2d\n", i, j);

    return 0;
}

In the beginning, i = 10, j = 20
At the end, i = 20, j = 10
/* swap happens in a function, with only values passed */

#include <stdio.h>

void swap(int n, int m);

int main(void)
{
   int i = 10, j = 20;

   printf("In the beginning, i = %2d, j = %2d\n", i, j);

   swap(i, j);

   printf("At the end, i = %2d, j = %2d\n", i, j);

   return 0;
}

void swap(int x, int y)    /* arguments passed "by value" */
{
   int temp;

   temp = x;
   x = y;
   y = temp;

   return;
}

In the beginning, i = 10, j = 20
At the end, i = 10, j = 20
swap happens in a function, with addresses passed

#include <stdio.h>

void swap(int *n, int *m);

int main(void)
{
    int i = 10, j = 20;

    printf("In the beginning, i = %2d, j = %2d\n", i, j);

    swap(&i, &j);

    printf("At the end, i = %2d, j = %2d\n", i, j);

    return 0;
}

void swap(int *x, int *y) /* arguments passed as addresses */
{
    int temp;

    temp = *x;
    *x = *y;
    *y = temp;

    return;
}

In the beginning, i = 10, j = 20
At the end, i = 20, j = 10