EENG212 – Algorithms & Data Structures
Fall 07/08 – Lecture Notes # 3

OUTLINE
♦ Review of Pointers in C
♦ Pointer Operators
♦ Passing Pointers to Functions
♦ Binary Search using Pointer notation

POINTERS
♦ Pointers are variables that contain memory addresses as their values.
  - A variable directly references a value.

<table>
<thead>
<tr>
<th>Ex:</th>
<th>int a , b= 15 ,c ,d ;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a = 7;</td>
</tr>
<tr>
<td></td>
<td>c = a + 3;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>100100</td>
<td>7</td>
</tr>
<tr>
<td>100101</td>
<td>15</td>
</tr>
<tr>
<td>100102</td>
<td>10</td>
</tr>
<tr>
<td>100103</td>
<td></td>
</tr>
<tr>
<td>100104</td>
<td></td>
</tr>
<tr>
<td>100105</td>
<td></td>
</tr>
</tbody>
</table>

- A pointer indirectly references a value. Referencing a value through pointer is known as indirection.

<table>
<thead>
<tr>
<th>Ex:</th>
<th>int *aptr, *z, * k ;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/*declaration of pointers */</td>
</tr>
<tr>
<td></td>
<td>float *m, *nptr;</td>
</tr>
</tbody>
</table>

Note: - aptr is a pointer to an integer. Or aptr points an object of integer type.
  - Pointer variables z and k are also pointers to integer.
  - M and nptr are pointers to float.

POINTER OPERATORS
1. The address operator, &: returns the address of its operand.

<table>
<thead>
<tr>
<th>Ex:</th>
<th>int a = 9, b;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>int *aptr, *bptr;</td>
</tr>
<tr>
<td></td>
<td>aptr = &amp;a;</td>
</tr>
<tr>
<td></td>
<td>bptr = &amp;b;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>100100</td>
<td>9</td>
</tr>
<tr>
<td>100101</td>
<td>a</td>
</tr>
<tr>
<td>100102</td>
<td>b</td>
</tr>
<tr>
<td>100103</td>
<td></td>
</tr>
<tr>
<td>100104</td>
<td></td>
</tr>
<tr>
<td>100105</td>
<td></td>
</tr>
</tbody>
</table>

Note: Here aptr is said to point to a;

2. The indirection/ dereferencing operator, *: returns the value of the object pointed by its operand.

int x=8, *xptr;
float y, *yptr;
y = 3.4;
yptr = &y;
xptr = &x;

printf(“The value of x = %d”, *xptr);
printf(“The value of y =%.2f”, *yptr);
Ex: The following program prints the values of the variables a and b, as well as the address of the variables (pointers to the variables) a and b. Run the program to see the output.

```
#include <stdio.h>
int main()
{
    int a=3;
    int b=1;
    int *aPtr; /*aPtr is a pointer to an integer */
    int *bPtr; /* aPtr is a pointer to an integer */
    aPtr = &a; /* aPtr is set to the address of a */
    b = *aPtr; /* the value of a is assigned to v */
    bPtr =&b; /* the address of b is assigned to bPtr */

    printf("a=%d &a=%p aPtr=%p *aPtr=%d", a, &a, aPtr, *aPtr);
    printf("b=%d &b=%p bPtr=%p *bPtr=%d", b, &b, bPtr, *bPtr);
    return 0;
}
```

---

**PASSING POINTERS TO FUNCTIONS**

- Pointers are very useful to simulate the *call by reference*. Instead of passing a copy of variable to the function (*call by value*), you can use pointers to pass address of a variable to a function (*call by reference*).

*Ex:* Here is a simple C program that illustrates the difference between ordinary arguments, which are passed by value, and pointer arguments which are passed by reference. Run the program to see the output.

```
#include <stdio.h>
void call_by_value(int , int );
void call_by_reference(int *, int *);

int main()
{
    int a=1;
    int b=3;

    printf("Before calling call_by_value: a=%d b=%d",a,b);
    call_by_value(a,b);
    printf("After calling call_by_value: a=%d b=%d",a,b);

    printf("Before calling call_by_reference: a=%d b=%d",a,b);
    call_by_reference(&a,&b);
    printf("After calling call_by_reference: a=%d b=%d",a,b);

    return 0;
}
```
void call_by_value(int x, int y)
{
    x=0;
    y=0;
    printf ("\n Within function call_by_value: a=%d b=%d", x, y);
}

void call_by_reference(int *xPtr, int *yPtr)
{
    *xPtr=0;
    *yPtr=0;
    printf ("\n Within function call_by_reference: *xPtr =%d *yPtr =%d", *xPtr, *yPtr);
}

**Binary Search**: Given a sorted array Binary Search algorithm can be used to perform fast searching of a search key on the sorted array.

The following program uses pointer notation to implement the binary search algorithm for the search key entered by the user in the following array:

(3, 5, 9, 11, 15, 17, 22, 25, 37, 68)

```c
#include <stdio.h>
define SIZE 10
int BinarySearch(int *, int);
int main()
{
    int a[SIZE] = {3, 5, 9, 11, 15, 17, 22, 25, 37, 68}, key, pos;
    printf("Enter the Search Key\n");
    scanf("%d", &key);
    pos = BinarySearch(a, key);
    if(pos == -1)
        printf("The search key is not in the array\n");
    else
        printf("The search key %d is at location \%d\n", key, pos);
    return 0;
}

int BinarySearch (int *aptr, int skey)
{
    int low=0, high=SIZE-1, middle;
    while(low <= high)
    {
        middle= (low+high)/2;
        if(skey == *(aptr+middle))
            return middle;
        else if(skey < *(aptr+middle))
            high = middle-1;
        else
            low = middle+1;
    }
    return -1;
}
```