Programming Language Lab

Bibhas Adhikari

IIT Kharagpur

October 22, 2020

Pointers

- *Pointers* are powerful structures to work with variables, functions, and data structures through their memory addresses
- *Pointers* are variables that contain a memory address as their value. To be specific, a pointer variable contains a memory address that points to another variable
  - Let $ivar$ be an integer variable that contains the value 25 with a memory address of $0x948312$
  - Let $ipointer$ be a pointer variable that does not contain a data value, but instead contains a memory address of $0x948312$, which is the same memory address of $ivar$
  - This means that the pointer variable $ipointer$ indirectly points to the value of 25
  - This entire concept is known as *indirection*

Pointers: declaration

$int\ x = 0;$
$int\ ivar = 25;$
$int\ *ptvar;$

- Writing the indirection operator ($*$) in front of the variable name is to declare a pointer. However, the pointer variable has not been assigned any value but it should be an integer data type. To refer a value through a pointer, we must assign an address to the pointer as follows:
  $ptvar = \&ivar;$
- Now a memory address of the $ivar$ variable to the pointer variable is assigned
- This is accomplished by placing the unary operator (&) in front of the variable $ivar$
- The operator (&) is referred to as the "address of" $ivar$

Pointers: initialization

- Pointer variables should be initialized with another variable's memory address, with 0, or with the keyword *NULL*

```
int *ptr1;
int *ptr2;
int *ptr3;
ptr1 = &x;
ptr2 = 0;
ptr3 = NULL;
```

Printing point variable values

print the memory address of pointers and non-pointer variables using the %p conversion specifier:

```
int x = 1;
int *iPtr;
iPtr = &x;
*iPtr = 5;
printf ("\n *iPtr = %p\n&x = %p\n", iPtr, &x);
```

Hexadecimal Number System and Addressing Memory

A number system based on 16 values (called base 16), which is explained in this appendix. Uses the 16 numerals $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E$, and F. Hex numbers are often followed by a lowercase h to indicate they are in hex (example: 78h)

Decimal	Hex	Binary	Decimal	Hex	Binary	Decimal	Hex	Binary
0	0	0	14	E	1110	28	1C	11100
1	1	1	15	F	1111	29	1D	11101
2	2	10	16	10	10000	30	1E	11110
3	3	11	17	11	10001	31	1F	11111
4	4	100	18	12	10010	32	20	100000
5	5	101	19	13	10011	33	21	100001
6	6	110	20	14	10100	34	22	100010
7	7	111	21	15	10101	35	23	100011
8	8	1000	22	16	10110	36	24	100100
9	9	1001	23	17	10111	37	25	100101
10	A	1010	24	18	11000	38	26	100110
11	B	1011	25	19	11001	39	27	100111
12	C	1100	26	1A	11010	40	28	101000
13	D	1101	27	1B	11011			

Passing arguments of functions via reference

- Recall that arguments of a function are passed by a value, as we discussed before
- However, this may not be efficient due to storage requirement of the value, consider the following example

```
int addTwoNumbers(int, int);
int x = 0;
int y = 0;
printf("\n Enter first number:  ");
scanf("%d",&x);
printf("\n Enter second number:  ");
scanf("%d",&y);
printf ("\nResult is %d\n", addTwoNumbers(x,y));
```

Example: passing an argument by reference

```
main()
{
int x = 0;
printf("\nEnter a number:  ");
scanf("%d",&x);
PassByValue(x);
printf("\nThe original value of x did not change: %d\n",x);
}
void PassByValue(int x)
{
x+ = 5;
printf("\nThe value of x is: %d\n",x);
}
```

Example: passing an argument by reference

```
main()
{
int x = 0;
printf("\nEnter a number:  ");
scanf("%d",&x);
PassByReference(&x);
printf("\nThe original value of x is: %d\n",x);
}
void PassByReference(int *ptrX)
{
*ptrX += 5;
printf("\nThe value of x is now: %d\n", *ptrX);
}
```

Passing arrays to functions

- Passing an array name to a pointer means assigning the first memory location of the array to the pointer variable
- In the following example, we create and initialize an array of 5 elements and declare a pointer that is initialized to the array name. Then initializing a pointer to an array name stores the first address of the array in the pointer

```
main()
{
int iArray[5] = {1,2,3,4,5};
int *iPtr = iArray;
printf("\nAddress of pointer: %p\n", iPtr);
printf("First address of array: %pn",&iArray[0]);
printf("\nPointer points to: %d\n", *iPtr);
printf("First element of array contains: %d\n", iArray[0]);
}
```

Example: passing array to a function

Passing a character array to a function to calculate the length of a the string:

```
main()
{
char aName[20] = {'\0'};
printf("\nEnter your first name: ");
scanf("%s", aName);
printf("\nYour first name contains ");
printf("%d characters\n", nameLength(aName));
}
int nameLength(char name[])
{
int x = 0;
while ( name[x] != '\0')
x++;
return x;
}
```

Example: passing arrays by reference

Modifying array using pass by reference techniques:

```
 main()
{
int x;
int iNumbers[3] = {2,4,6};
printf("\nThe current array values are: ");
for (x = 0; x < 3; x++)
printf("%d", iNumbers[x]);
printf("\n");
squareNumbers(iNumbers);
printf("\nThe modified array values are:  ");
for (x = 0; x < 3; x++)
printf("%d", iNumbers[x]);
printf("\n");
}
```

Example continued

```
void squareNumbers(int num[])
{
int x;
for (x = 0; x < 3; x++)
num[x] = num[x] * num[x];
}
```