To:
Tenouk

C LAB WORKSHEET 15b
C Pointers, Arrays, Functions, struct Part 3

1. More questions, answers and exercises on C & C++ pointers, indirection operators etc.
2. The dot and the arrow operators used to access pointers members.
3. Tutorial references that should be used together with this worksheet are: arrays 1, array 2, pointers 1, pointer 2, pointer 3, functions 1 , function 2, function 3, function 4 and structs.

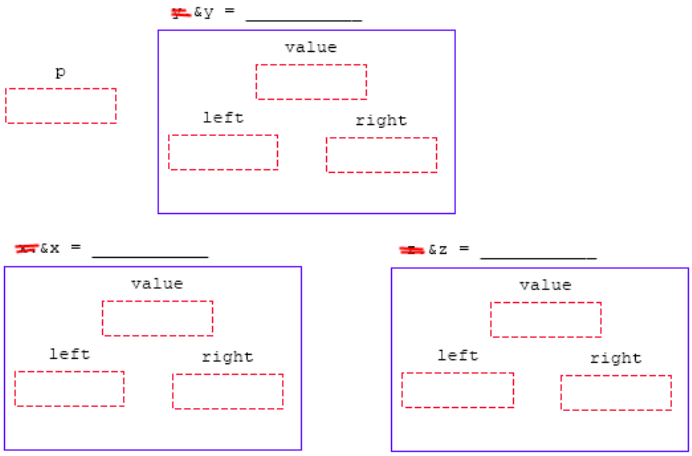
10. Show the output and answer the questions for the following exercise.

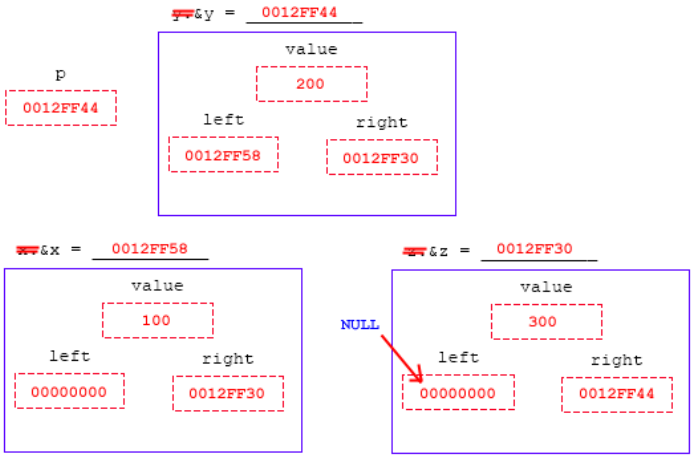
```
#include <stdio.h>

// struct declaration...
struct Branch
{
    int value;
    struct Branch *left;
    struct Branch *right;
};

void main(void)
{

struct Branch x, y, z, *p = &y;

printf("&x = %p, &y = %p, &z = %p, p = %p\n", &x, &y, &z, p);
x.value = 100;
y.value = 200;
z.value = 300;

x.left = (struct Branch *) 0;
x.right = &z;
z.left = NULL;
z.right = &y;
y.left = &x;
y.right = &z;
printf("\n&x = %p: x.value = %d, x.left = %p, x.right = %p\n", &x, x.value, x.left, x.right);
printf("&y = %p: y.value = %d, y.left = %p, y.right = %p\n", &y, y.value, y.left, y.right);
printf("&z = %p: z.value = %d, z.left = %p, z.right = %p\n", &z, z.value, z.left, z.right);
}
```

```
&x = 0012FF58, &y = 0012FF44, &z = 0012FF30, p = 0012FF44

&x = 0012FF58: x.value = 100, x.left = 00000000, x.right = 0012FF30
&y = 0012FF44: y.value = 200, y.left = 0012FF58, y.right = 0012FF30
&z = 0012FF30: z.value = 300, z.left = 00000000, z.right = 0012FF44
Press any key to continue . . .
```

a. How many items each do x, y and z hold?
b. For each item that they hold, how many are pointers? And pointers to what data type?
c. How many items does p hold? It can store addresses to what data type? What is its value?
d. Where are the variables x, y and z stored? Give their addresses.
e. Fill in all the addresses and boxes in the following Figures.

a. Based on the struct member variables, each x, y and z hold three items that is value, *left and *right.
b. Two are pointers (*left and *right) of struct Branch data type.
c. p also hold three items and it can store addresses of struct Branch. From the output sample shown below, its value is 0012FF44, same as the address of y.
d. From the output sample shown below &x = 0012FF58, &y = 0012FF44 and &z = 0012FF30.
e. See the following Figure.





11. For the following and the next exercises, add the some code at the end of the code given in exercise 10. Show the output and answer the questions based on the added code.

```
#include <stdio.h>

// struct declaration...
struct Branch
{
    int value;
    struct Branch *left;
    struct Branch *right;
};

 void main(void)
{
  struct Branch x, y, z, *p = &y;

  printf("&x = %p, &y = %p, &z = %p, p = %p\n", &x, &y, &z, p);
  x.value = 100;
  y.value = 200;
  z.value = 300;

  x.left = (struct Branch *) 0;
  x.right = &z;
  z.left = NULL;
  z.right = &y;
  y.left = &x;
  y.right = &z;
  printf("\n&x = %p: x.value = %d,\nx.left = %p, x.right = %p\n", &x, x.value, x.left, x.
right);
  printf("&y = %p: y.value = %d,\ny.left = %p, y.right = %p\n", &y, y.value, y.left, y.
right);
  printf("&z = %p: z.value = %d,\nz.left = %p, z.right = %p\n\n", &z, z.value, z.left, z.
right);

  printf("&y = %p, y.value = %d \n", &y, y.value);
  printf("y.left = %p, y.right = %p \n", y.left, y.right);
  printf("p = %p, (*p).value = %d \n", p, (*p).value);
  printf("(*p).left = %p, (*p).right = %p\n", (*p).left, (*p).right);
}
```

```
&x = 0012FF58, &y = 0012FF44, &z = 0012FF30, p = 0012FF44

&x = 0012FF58: x.value = 100,
x.left = 00000000, x.right = 0012FF30
&y = 0012FF44: y.value = 200,
y.left = 0012FF58, y.right = 0012FF30
&z = 0012FF30: z.value = 300,
z.left = 00000000, z.right = 0012FF44

&y = 0012FF44, y.value = 200
y.left = 0012FF58, y.right = 0012FF30
p = 0012FF44, (*p).value = 200
(*p).left = 0012FF58, (*p).right = 0012FF30
Press any key to continue . . .
```

a. When printing y.value, why was a %d used for the format string?
b. When printing y.left, why was a %p used for the format string?
c. What is the address in p? This is the address of which variable?
d. Can you print p.value?
e. Why do you need an asterisk in front of the p when printing (*p).value?
f. In the three places that this appears, "(*p).", change it to: "p →" (a combination of hyphen and a greater than operator). Use no period or spaces. Is there any difference in the expressions? For instance, try the following code.

```
printf("p = %p, p→value = %d \n", p, p→value);
```

g. (*p).left gives the address stored in the left member of which variable?
h. Why does (*p).left give the address stored in the left member of y?
i. The address stored in (*p).left is the address of which variable?

a. The %d format string used because value variable is an integer data type.
b. The %p format string used because left variable is a pointer variable of struct Branch data type.
c. Based on the sample output, &p = 0012FF44 and this is the address of y variable.
d. No, p.value cannot be printed because p is a pointer variable.
e. This is because p is a pointer that hold an address instead of the value. By using the asterisk, we mean to print the value.
f. There is no difference.
g. y variable.
h. This is because of the *p = &y.
i. x variable.

j. Now add the following code to the end of your program. Rebuild and show the output for the added code.

```
p = (*p).left;
printf("\np = %p, (*p).value = %d \n", p, (*p).value);
printf("(*p).left = %p, (*p).right = %p\n", (*p).left, (*p).right);
```

k. Now p has the address of which variable?

```
&x = 0012FF58, &y = 0012FF44, &z = 0012FF30, p = 0012FF44

&x = 0012FF58: x.value = 100,
x.left = 00000000, x.right = 0012FF30
&y = 0012FF44: y.value = 200,
y.left = 0012FF58, y.right = 0012FF30
&z = 0012FF30: z.value = 300,
z.left = 00000000, z.right = 0012FF44

&y = 0012FF44, y.value = 200
y.left = 0012FF58, y.right = 0012FF30
p = 0012FF44, p->value = 200
p->left = 0012FF58, p->right = 0012FF30

p = 0012FF58, (*p).value = 100
(*p).left = 00000000, (*p).right = 0012FF30
Press any key to continue . . .
```

k. p has the address of x variable.

l. Convert these lines to the notation that uses the arrow (→). Is the result the same?

```
p = p->left;
printf("\np = %p, p->value = %d \n", p, p->value);
printf("p->left = %p, p->right = %p\n", p->left, p->right);
```

Yes, the result is same.

m. Now add the following code at the end of your previous program. Rebuild and show the output for the added code.

```
p = (*p).right;
printf("\np = %p, (*p).value = %d \n", p, (*p).value);
printf("(*p).left = %p, (*p).right = %p\n", (*p).left, (*p).right);
```

```
&x = 0012FF58, &y = 0012FF44, &z = 0012FF30, p = 0012FF44

&x = 0012FF58: x.value = 100,
x.left = 00000000, x.right = 0012FF30
&y = 0012FF44: y.value = 200,
y.left = 0012FF58, y.right = 0012FF30
&z = 0012FF30: z.value = 300,
z.left = 00000000, z.right = 0012FF44

&y = 0012FF44, y.value = 200
y.left = 0012FF58, y.right = 0012FF30
p = 0012FF44, p->value = 200
p->left = 0012FF58, p->right = 0012FF30

p = 0012FF58, p->value = 100
p->left = 00000000, p->right = 0012FF30

p = 0012FF30, (*p).value = 300
(*p).left = 00000000, (*p).right = 0012FF44
Press any key to continue . . .
```

n. Convert these three lines to the arrow notation (->) as done previously and test your code.

```
p = p->right;
printf("\np = %p, p->value = %d\n", p, p->value);
printf("p->left = %p, p->right = %p\n", p->left, p->right);
```

n. See the following Figure.

```
&x = 0012FF58, &y = 0012FF44, &z = 0012FF30, p = 0012FF44

&x = 0012FF58: x.value = 100,
x.left = 00000000, x.right = 0012FF30
&y = 0012FF44: y.value = 200,
y.left = 0012FF58, y.right = 0012FF30
&z = 0012FF30: z.value = 300,
z.left = 00000000, z.right = 0012FF44

&y = 0012FF44, y.value = 200
y.left = 0012FF58, y.right = 0012FF30
p = 0012FF44, p->value = 200
p->left = 0012FF58, p->right = 0012FF30

p = 0012FF58, p->value = 100
p->left = 00000000, p->right = 0012FF30

p = 0012FF30, p->value = 300
p->left = 00000000, p->right = 0012FF44
Press any key to continue . . .
```

o. Now the address stored in p is the address of which variable?

o. The address stored in p is the address of z variable.

p. Now add the following code to the end of your program that you have so far.

```
p = (*p).right;
printf("\np = %p, (*p).value = %d \n", p, (*p).value);
printf("(*p).left = %p, (*p).right = %p\n", (*p).left, (*p).right);
```

```
&x = 0012FF58, &y = 0012FF44, &z = 0012FF30, p = 0012FF44

&x = 0012FF58: x.value = 100,
x.left = 00000000, x.right = 0012FF30
&y = 0012FF44: y.value = 200,
y.left = 0012FF58, y.right = 0012FF30
&z = 0012FF30: z.value = 300,
z.left = 00000000, z.right = 0012FF44

&y = 0012FF44, y.value = 200
y.left = 0012FF58, y.right = 0012FF30
p = 0012FF44, p->value = 200
p->left = 0012FF58, p->right = 0012FF30

p = 0012FF58, p->value = 100
p->left = 00000000, p->right = 0012FF30

p = 0012FF30, p->value = 300
p->left = 00000000, p->right = 0012FF44

p = 0012FF44, (*p).value = 200
(*p).left = 0012FF58, (*p).right = 0012FF30
Press any key to continue . . .
```

q. Convert these lines to the arrow notation as done previously and test your code. Is there an address or a member name on the left side of the arrow? What about on the right side?

```
p = p->right;
printf("\np = %p, p->value = %d \n", p, p->value);
printf("p->left = %p, p->right = %p\n", p->left, p->right);
```

r. Now the address that is stored in p is the address of which variable?

```
&x = 0012FF58, &y = 0012FF44, &z = 0012FF30, p = 0012FF44

&x = 0012FF58: x.value = 100,
x.left = 00000000, x.right = 0012FF30
&y = 0012FF44: y.value = 200,
y.left = 0012FF58, y.right = 0012FF30
&z = 0012FF30: z.value = 300,
z.left = 00000000, z.right = 0012FF44

&y = 0012FF44, y.value = 200
y.left = 0012FF58, y.right = 0012FF30
p = 0012FF44, p->value = 200
p->left = 0012FF58, p->right = 0012FF30

p = 0012FF58, p->value = 100
p->left = 00000000, p->right = 0012FF30

p = 0012FF30, p->value = 300
p->left = 00000000, p->right = 0012FF44

p = 0012FF44, p->value = 200
p->left = 0012FF58, p->right = 0012FF30
Press any key to continue . . .
```

q. On the left-side is an address and on the right-side is a member name.

r. The address that stored in p is the address of y variable.

r. Can you do any of these assignments? Why or why not?

```
p = (*p).value;
p = &p;
y.left = &y;
```

p = (*p).value; - Cannot. value variable holds an integer instead of an address.
p = &p; - Cannot. p already a pointer that holds an address.
y.left = &y; - Can. Both side have similar data type, a pointer to an address.

More Questions And Answers

For all questions and their parts, start with the following code definitions.

```
#include <stdio.h>

// struct declaration...
struct Ball
{
      int Radius;
      struct Ball *last;
      struct Ball *next;
};

void main(void)
{
    struct Ball x, y, *p5 = &x, *p6 = &y;

    int i = 0, m[5] = {3, 6, 9, 12, 15}, *p1, *p3, *q[5];
    double j, a[5] = {20.0, 30.5, 40.3, 50.3, 60.4}, *p2 = a;

    // ...
    // ...
    // ...
}
```

12. Show the contents of the variables that have changed after executing each of the following codes.

    a.
    ```
    p1 = &i;
    *p1 = 8;
    ```

    b.
    ```
    p1 = &m[0];
    p1 = p1 + 2;
    *p1 = *p1 + 8;
    ```

    c.
    ```
    p1 = &i;
    p3 = m;
    *(p3 + 1) = *p3 + *p1;
    ```

    d.
    ```
    p1 = m + 2;
    p3 = p1 + 1;
    i = *p1 + *p3;
    ```

    e.
    ```
    for(i = 0; i <= 4; ++i)
            q[i] = m + i;
    for(i = 0; i <= 4; ++i)
            *(a + i) = *q[i] + a[i];
    ```

    f.
    ```
    for(i = 4; i >= 0; --i)
    {
            q[i] = &m[4 – i];
            *p2 = *q[i] + *(a + i);
    }
    ```

We add some code so that the output and the changed variables can be seen.

```
p1 = &i;
printf("p1 = %p, *p1 = %d\n",p1, *p1);
*p1 = 8;
printf("p1 = %p, *p1 = %d\n",p1, *p1);
```

```
p1 = 0012FF20, *p1 = 0
p1 = 0012FF20, *p1 = 8
Press any key to continue .
```

```
p1 = &m[0];
printf("p1 = %p\n",p1);
p1 = p1 + 2;
printf("p1 = %p\n",p1);
*p1 = *p1 + 8;
printf("*p1 = %d\n",*p1);
```

```
p1 = 0012FF04
p1 = 0012FF0C
*p1 = 17
Press any key to continue
```

```
p1 = &i;
printf("p1 = %p\n",p1);
p3 = m;
printf("p3 = %p\n",p3);
*(p3 + 1) = *p3 + *p1;
printf("*p1 = %d, p3 = %p, *p3 = %d\n",*p1,p3, *p3);
```

```
p1 = 0012FF20
p3 = 0012FF04
*p1 = 0, p3 = 0012FF04, *p3 = 3
Press any key to continue . . .
```

```
p1 = m + 2;
printf("p1 = %p, m = %p\n",p1, m);
p3 = p1 + 1;
printf("p3 = %p, p1 = %p\n",p3, p1);
i = *p1 + *p3;
printf("i = %d, *p1 = %d, *p3 = %d\n",i,*p1, *p3);
```

```
p1 = 0012FF0C, m = 0012FF04
p3 = 0012FF10, p1 = 0012FF0C
i = 21, *p1 = 9, *p3 = 12
Press any key to continue . .
```

```
for(i = 0; i <= 4; ++i)
{
   q[i] = m + i;
   printf("m = %p, i = %d, q[%d] = %p\n", m, i, i, q[i]);
}
printf("\n");
for(i = 0; i <= 4; ++i)
{
   *(a + i) = *q[i] + a[i];
   printf("*q[%d] = %d, a[%d] = %.2f, a = %p, i = %d\n", i, *q[i],i,a[i],a, i);
}
```

```
m = 0012FF04, i = 0, q[0] = 0012FF04
m = 0012FF04, i = 1, q[1] = 0012FF08
m = 0012FF04, i = 2, q[2] = 0012FF0C
m = 0012FF04, i = 3, q[3] = 0012FF10
m = 0012FF04, i = 4, q[4] = 0012FF14

*q[0] = 3, a[0] = 23.00, a = 0012FE90, i = 0
*q[1] = 6, a[1] = 36.50, a = 0012FE90, i = 1
*q[2] = 9, a[2] = 49.30, a = 0012FE90, i = 2
*q[3] = 12, a[3] = 62.30, a = 0012FE90, i = 3
*q[4] = 15, a[4] = 75.40, a = 0012FE90, i = 4
Press any key to continue . . .
```

```
for(i = 4; i >= 0; --i)
{
   q[i]=&m[4-i];
```

```
    printf("i = %d, &m = %p, &m[4-%d] = %p\n", i, &m, i, &m[4-i]);
    *p2 = *q[i] + *(a + i);
    printf("a = %p, i = %d, *(a+%d) = %.2f, *q[%d] = %d, *p2=%.2f\n", a, i, i, *(a+i), i, *q[i],
*p2);
}
```

```
i = 4, &m = 0012FF04, &m[4-4] = 0012FF04
a = 0012FE90, i = 4, *(a+4) = 60.40, *q[4] = 3, *p2=63.40
i = 3, &m = 0012FF04, &m[4-3] = 0012FF08
a = 0012FE90, i = 3, *(a+3) = 50.30, *q[3] = 6, *p2=56.30
i = 2, &m = 0012FF04, &m[4-2] = 0012FF0C
a = 0012FE90, i = 2, *(a+2) = 40.30, *q[2] = 9, *p2=49.30
i = 1, &m = 0012FF04, &m[4-1] = 0012FF10
a = 0012FE90, i = 1, *(a+1) = 30.50, *q[1] = 12, *p2=42.50
i = 0, &m = 0012FF04, &m[4-0] = 0012FF14
a = 0012FE90, i = 0, *(a+0) = 57.50, *q[0] = 15, *p2=57.50
Press any key to continue . . .
```

13. Which of the following are valid?

a. last = next;
b. x = y;
c. p5 = p6;
d. x = 10;
e. x.last = 10;
f. x.Radius = 10;
g. p5.Radius = 10;
h. p5 -> Radius = 10;
i. (*x).Radius = 10;
j. *(x.Radius) = 10;
k. *(x.last) = &y;
l. (*x).last = &y;
m. (*p5).last = &y;
n. *(p5.last) = &y;
o. p5 -> last = &y;
p. *p5 = y;
q. p5 -> last = NULL;
r. p5 -> next = 10;

a. last = next; - Invalid, we cannot directly access the member variable.
b. x = y;
c. p5 = p6;
d. x = 10; - Invalid, x is a variable of type struct Ball and should be written such as x. Radius = 10 or on the right-side must be struct Ball type such as x = y
e. x.last = 10; - Invalid, last is a pointer variable. On the right-side must be a struct Ball type instead of an integer.
f. x.Radius = 10;
g. p5.Radius = 10; - Invalid, p5 is a pointer variable of struct Ball type so must be written as (*p5).Radius = 10 or p5->Radius = 10.
h. p5 -> Radius = 10;
i. (*x).Radius = 10; - Invalid, Radius is an integer so must be something like x. Radius = 10
j. *(x.Radius) = 10; - Invalid, Radius is an integer so must be something like x. Radius = 10
k. *(x.last) = &y; - Invalid, supposed to be written as x.last = &y;
l. (*x).last = &y; - Invalid, supposed to be written as x.last = &y;
m. (*p5).last = &y;
n. *(p5.last) = &y; - Invalid, supposed to be written as (*p5).last = &y; or p5->last = &y;
o. p5 -> last = &y;
p. *p5 = y;
q. p5 -> last = NULL;
r. p5 -> next = 10; - Invalid, p5 is a pointer variable of struct Ball type and last is a pointer variable so on the right-side it must be an address

Do A Programming

1. Write a function called AddTwo(). It will receive three pointers to floats. It will add the numbers pointed to by the first two floats and store their sum in the location pointed to by the third pointer. In main(), define j, k and m. Scan in j and k, send their addresses to AddTwo() and have main() print the address of m and its value that is in it from AddTwo().

```
#include <stdio.h>

float * AddTwo(float *, float *, float *);

void main()
{
   // we need to initialize pointer variables,
   // giving some dummy values
   float p =1.2f, q=2.2f, r=3.3f, *j=&p, *k=&q, *m=&r, *n;

   printf("Enter two floats: \n");
   scanf_s("%f%f", j,k,sizeof(float*),sizeof(float*));
   n = AddTwo(j,k,m);
   printf("The sum of %.2f and %.2f is %.2f\nwhich stored at %p.\n", *j, *k, *n, n);
}

float * AddTwo(float * pointTo_j, float * pointTo_k, float * pointTo_m)
{
   *pointTo_m = *pointTo_j + *pointTo_k;
   printf("The sum is %.2f\n", *pointTo_m);
   return pointTo_m;
}
```

```
Enter two floats:
7.72 2.28
The sum is 10.00
The sum of 7.72 and 2.28 is 10.00
which stored at 0012FF48.
Press any key to continue . . .
```

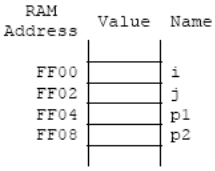
2. Define an integer array called a[10]. Initialize it to some values. Then assign the address of the slot whose index is 4 to an integer pointer called p. Print the contents of the array and the address of each slot using the name a but no brackets. Now do the same using the name p.

```
#include <stdio.h>

void main()
{
   int i, a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9 , 10}, *p;

   p = &a[3];

   for(i=0;i<=9;i++)
   {
      printf("a[%d] = %d at %p\n", i, a[i], a + i);
   }
   printf("\n");
   for(i=0;i<=9;i++)
   {
      printf("a[%d] = %d at %p\n", i, a[i], p - 3 + i);
   }
}
```

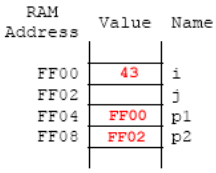
```
a[0] = 1 at 0012FF30
a[1] = 2 at 0012FF34
a[2] = 3 at 0012FF38
a[3] = 4 at 0012FF3C
a[4] = 5 at 0012FF40
a[5] = 6 at 0012FF44
a[6] = 7 at 0012FF48
a[7] = 8 at 0012FF4C
a[8] = 9 at 0012FF50
a[9] = 10 at 0012FF54

a[0] = 1 at 0012FF30
a[1] = 2 at 0012FF34
a[2] = 3 at 0012FF38
a[3] = 4 at 0012FF3C
a[4] = 5 at 0012FF40
a[5] = 6 at 0012FF44
a[6] = 7 at 0012FF48
a[7] = 8 at 0012FF4C
a[8] = 9 at 0012FF50
a[9] = 10 at 0012FF54
Press any key to continue .
```

More Drills

1. The following diagram shows four variables and the addresses where they are stored. After the execution of these statements, show the values of the locations that are assigned.




```
int i = 43, j;
int *p1, *p2;
p1 = &i;
p2 = p1;
p2 = p2 + 1;
```



p1 gets the address of i, p2 gets p1 and p2 then gets incremented by one storage unit. p1 and p2 are pointers to integers.

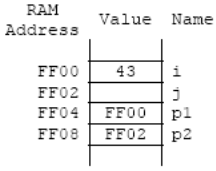
2. The asterisk (*) is the indirection operator. *p1 refers to the location in memory that has its address in p1. Or, in other words, the location whose address is given in p1. What will the following code print? Notice that p1 uses a %p and *p1 uses a %d. Differentiate between the memory address and the actual value stored at the address.

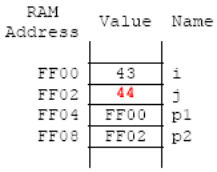
```
printf("p1 = %p, *p1 = %d\n", p1, *p1);
```

p1 simply prints the address in it. However *p1 prints the value stored at a location pointed by p1. *p1 prints 43 because the address in p1, namely FF00, has 43 stored in it.

3. Show the new contents of the locations in the diagram after the following code is executed.

```
*p2 = *p1;
j = j + 1;
```

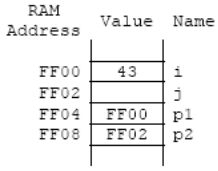


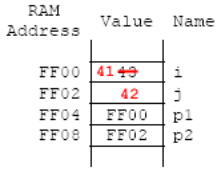


The contents that p1 points to are stored in the location that p2 points to. Then 1 is added to j.

4. After the following code execution, show the contents of the locations that will be changed.

```
*p2 = *(p1 + 1) - 2;
*p1 = *p1 - 2;
```

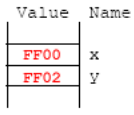




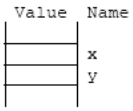
1 is added to p1, which is the address of FF02. From the contents of the location pointed to by FF02, 2 is subtracted, making j equal to 42. p1 remains the same. Then the contents pointed to by p1, which is actually i, has a 2 subtracted from it. now the current value of i is 41.

5. Continue from the previous drill, for the following code, show the values for x and y in the diagram. From main(), ShakeIt() is called. Show its effects in memory. Lastly, what is printed?

```
ShakeIt(p1, p2);
printf("*p1 = %d, *p2 = %d\n", *p1, *p2);

void ShakeIt(int *x, int *y)
{
  *x = 0;
  *y = 0;
}
```



The values for x and y are FF00 and FF02 respectively. *x is actually i and *y is actually j. These are set to zero in ShakeIt(). The two zeroes, the values of i and j are printed.



6. For the following code, show the printout. You may try to complete the code by adding the main() etc.

```
char x[10] = "Inside", y[10] = "and out";

ShakeIt(x, y);
printf("%s %s\n", x, y);

void ShakeIt(char *a, char *b)
{
  *(a + 1) = 'M';
  *b = 'u';
}
```

```
#include <stdio.h>

void ShakeIt(char *, char *);
void main()
{
  char x[10] = "Inside", y[10] = "and out";
  ShakeIt(x, y);
  printf("%s %s\n", x, y);
}

void ShakeIt(char *a, char *b)
{
  *(a + 1) = 'M';
  *b = 'u';
}
```

```
IMside und out
Press any key to continue . .
```

Array names are actually array addresses. hence, the pointer a actually has its address at the beginning of x[ ] . This is also the case with y[ ]. When ShakeIt() changes the contents of the location pointed to by a and b, that changes the array x[ ] and y[ ].

7. Are there any errors in the following code? If not, show the printout.

```
void ShakeIt(char *, char *);

void main(void)
{
  char x[10] = "Inside",
       y[10] = "and out";

  ShakeIt(&x[0], &y[0]);
  printf("%s %s\n", x, y);
}

void ShakeIt(char a[ ], char b[ ])
{
  a[1] = 'M';
  b[0] = 'u';
}
```

```
IMside und out
Press any key to continue .
```

No errors exist. Address can be passed to an array and the pointers can be used as arrays. The output is the same.

8. For the following questions, take a[ ] to be an array and specify which are true.

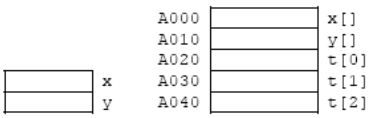
   a. a == &a[0]
   b. a + 1 == &a[1]
   c. *a == a[0]
   d. *(a + 1) == a[1]

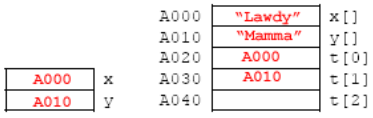
   a. a == &a[0] - True
   b. a + 1 == &a[1] - True
   c. *a == a[0] - True
   d. *(a + 1) == a[1] - True

All are true. a is the address of &a[0]. *a accesses what is stored in a[0], as does a[0].

9. Here we have an array of pointers. Show the values of the memory locations in the provided diagram.

```
char x[ ] = "Lawdy", y[ ] = "Mamma", *t[3];
t[0] = x;
t[1] = y;
```



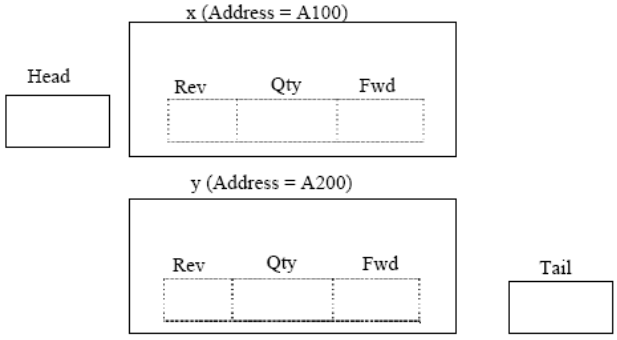


10. For the following scanf_s()/scanf(), "Hey" is read in. Show the contents of the memory locations after the execution of the given code.

```
t[2] = t[1];
strcpy_s(t[2], 10, t[0]);
scanf_s("%s", t[0], 10);
```

11. In the following structure, Rev and Fwd are pointers to type struct Node. Draw a diagram of each of the four variables. Which have three members and which have only one?

```
struct Node
{
  int Qty;
  struct Node *Rev;
  struct Node *Fwd;
};

struct Node x, y, *Head, *Tail;
```



x and y have three members but Head and Tail have one only (an address).

12. Which of the following statements are valid? (True or false).

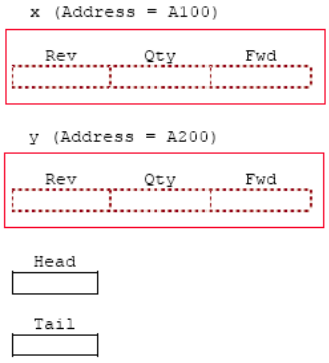
a. Head = &x;
b. x = &Head;
c. Head = Tail;
d. Head = &Tail;
e. x = &y;
f. x.Rev = 30;
g. x.Rev = &Head;
h. x.Rev = &y;
i. x.Rev = &x;
j. x.Qty = 30;
k. x.Qty = &y;
l. x = y;

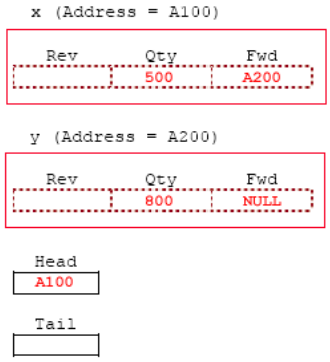
a. Head = &x; - Valid
b. x = &Head; - Not Valid
c. Head = Tail; - Valid
d. Head = &Tail; - Not Valid
e. x = &y; - Not Valid
f. x.Rev = 30; - Not Valid
g. x.Rev = &Head; - Not Valid
h. x.Rev = &y; - Valid
i. x.Rev = &x; - Valid
j. x.Qty = 30; - Valid
k. x.Qty = &y; - Not Valid
l. x = y; - Valid

Head can store only an address of a struct Node, so (a) and (c) are valid and (d) is not valid. x.Rev also can store only an address of a struct Node, so (h) and (i) are valid but (f) and (g) are not valid. In (l), we assign one entire structure to another with similar data type (struct Node) and it is valid.

13. In the diagram for solution 11 (shown below), fill in the values that the following code will assign.

```
Head = &x;
x.Fwd = &y;
// a NULL means that the address doesn't point to any object
// or pointing to address 0.
y.Fwd = NULL;
x.Qty = 500;
y.Qty = 800;
```





The address of x, which is A100, is stored in Head. The address of y, which is A200, is stored in x, Fwd and so on.

14. Write the statements to set up the following pointers: Have Tail point to y, have the Rev member of y point to x, and have the Rev member of x point to nothing.

```
Tail = &y;
y.Rev = &x;
x.Rev = NULL;
```

15. One way to access the Qty member of x is to state "x.Qty". Another way of doing that now is to state "(*Head).Qty". *Head will bring us to the location of A100 or x. And (*Head).Qty will bring us to the Qty member. A more common way of coding this is to state: "Head -> Qty". Give three ways to access the Fwd member of x.

1. x.Fwd
2. (*Head).Fwd
3. Head->Fwd. Could also be written as y.Rev->Fwd

16. Try writing the loop that starts with a pointer p, which is initialized to Head and then traverses the chain by following the Fwd pointers until it becomes NULL. Have the loop simply write out the values of p and the Qty members of each node as it traverses the chain. To advance the pointer p to the next link, p would have to be set equal to (*p).Fwd. Here is how your printout may look.

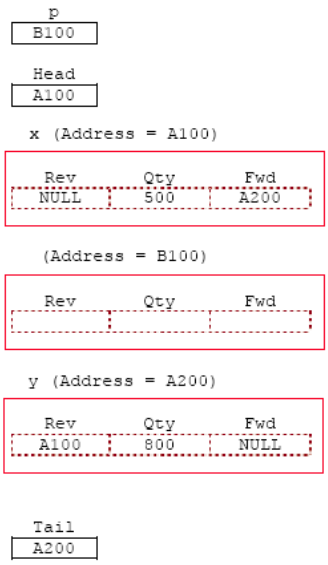
```
A100   500
A200   800
```

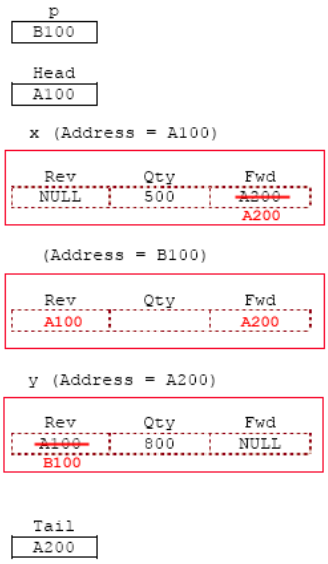
```
for(p = Head; p!=NULL; p = p->Fwd)
   printf("%p %d\n", p, p->Qty);
```

17. Now try reversing the traversal, starting p with Tail and following the Rev pointers.

```
for(p = Tail; p != NULL; p = p->Rev)
   printf("%p %d\n", p, p->Qty);
```

18. Using malloc(), we have created a new node that has no name, like x and y, but it has an address and it is stored in p. Let us pretend that the address stored in p is B100. How would you write the code so that this new node is placed between x and y. That is, x and y should point to the new node. Firstly, show the values in the diagram shown below that need to be changed.





19. There should be four addresses assigned or changed in the previous diagram. Give the four statements that will do that. Does the order of these statements matter? Do not use the pointers Head or Tail.

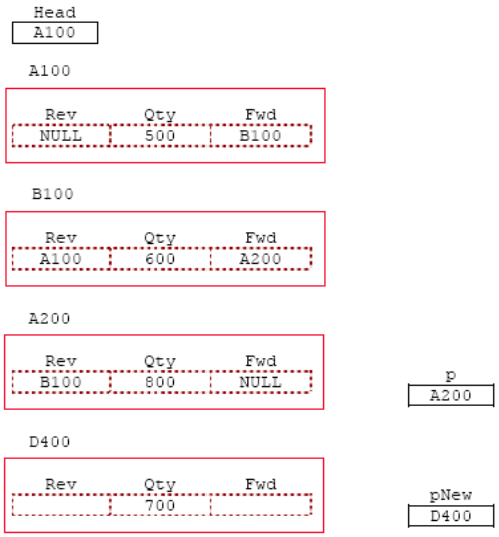
1. p->Rev = x.Fwd;
2. p-> Fwd = y.Rev;
3. x.Fwd = p;
4. y.Rev = p;

The first of the last two statements can be switched. However, the first two statements should come first. The Fwd and Rev members should be used first before new pointers are copied in them.

20. Now, do the same as above except do not use x or y. Instead, use Head and Tail. Let us also place a quantity, Qty of 600 in the new node.

```
p->Rev = Head->Fwd;
p->Fwd = Tail->Rev;
Head->Fwd = p;
Tail->Rev = p;
p->Qty = 600;
```

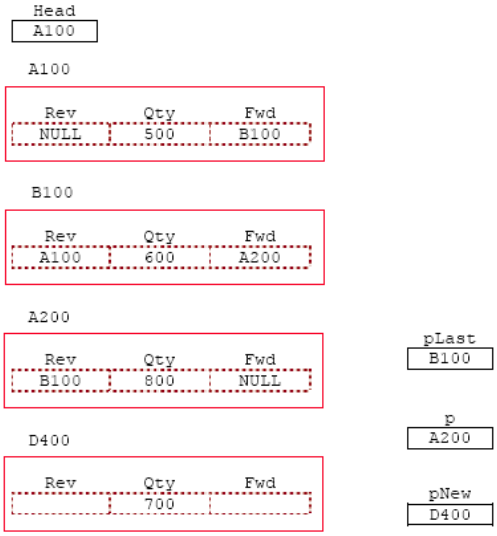
21. We will add a new node whose address is in pNew. Its quantity is 700 and we are to add it in the list. To start, first write the for loop that will start p at Head and make it follow the Fwd pointers until it finds a NULL. That is, p should follow A100, B100 and A200 where the NULL is found. Also fill in the diagram.



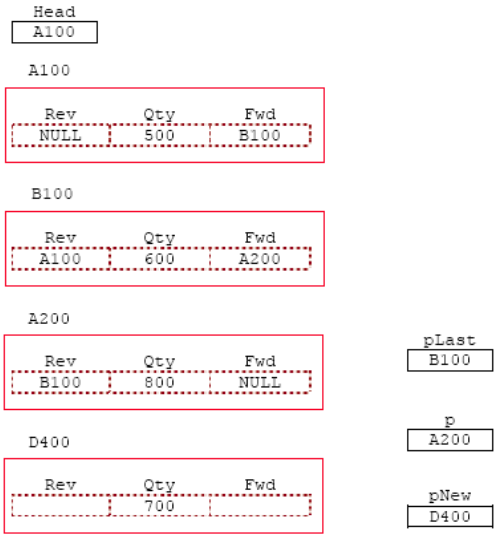

```
p = Head;
for( ; p != NULL; p = p->Fwd)
```

22. Now add an if statement to this loop that will break out of it when the quantity pointed to by p is greater than the quantity pointed to by pNew. In this example, the loop will break out when p becomes A200. That is, when p is A200 the 800 pointed to by p is greater than the 700 pointed to by pNew.

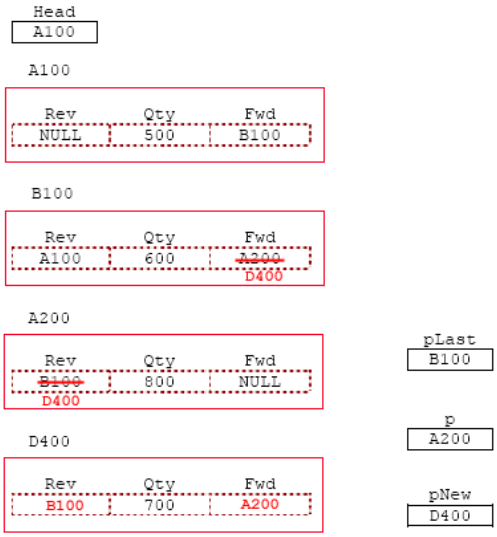
```
p = Head;
for( ; p != NULL; p = p->Fwd)
   if(p->Qty > pNew->Qty)
      break;
```



23. Since we know the Rev pointer from p, assign the address of the previous node from p into a pointer called pLast. That is, pLast should now be set to B100 after the loop. Then using pLast and p, change and assign the needed pointers so that the new node is linked between B100 and A200 in both directions. Also fill in the diagram for the changed and new address or/and data.




```
p = Head;
for( ; p != NULL; p = p->Fwd)
   if(p->Qty > pNew->Qty)
      break;
pLast = p->Rev;
pNew->Fwd = p;
pNew->Rev = pLast;
p->Rev = pNew;
pLast->Fwd = pNew;
```



24. Now assume that our node didn't have the Rev pointers. Rewrite the code so that the pointer pLast follows one node behind p inside the loop. When we need to link up the pointers, we will know where the previous node is. Your new code should make no reference to Rev pointers because now they are not needed.

```
p = Head;
for( ; p != NULL; p = p->Fwd)
   if(p->Qty > pNew->Qty)
      break;
   else
     pLast = p;
pNew->Fwd = p;
pLast->Fwd = pNew;
```

25. What changes are needed in the provided code of the drill 24 solution to accommodate each of the following special conditions?

```
p = Head;
for( ; p != NULL; p = p->Fwd)
  if(p->Qty > pNew->Qty)
       break;
  else
       pLast = p;
pNew -> Fwd = p;
pLast -> Fwd = pNew;
...
```

a. The new quantity is greater than the quantity in the last node of the list.
b. The new quantity is greater than the quantity in the first node of the list.
c. The linked list is empty; that is, to begin with, Head is NULL.

No changes are needed for part (a). Answers for (b) and (c) are as follows:

```
p = Head;
 for( ; p != NULL; p = p->Fwd)
   if(p->Qty > pNew->Qty)
        break;
   else
        pLast = p;
 pNew -> Fwd = p;

if(p == Head)
   Head = pNew;
else
   pLast->Fwd = pNew;
```

26. In C++ what standard library that will be used for linked list and other related data manipulation of the C++ constructs?

Standard Template Library (STL).

27. Which of the following are advantages of a double linked list and which are advantageous of a single linked list?

a. Take less memory space.
b. Easier to use if you are dumped in the middle of the linked list and need to go back one node.
c. Easier to traverse in both directions.
d. Less pointer maintenance.

a. Single.
b. Double.
c. Double.
d. Single.

www.tenouk.com

To: Tenouk tenouk.com, 2008