To:
Tenouk

C LAB WORKSHEET 15a
C Pointers, Arrays, Functions, struct Part 2

1. More working program examples, questions and practices on link and linked list.
2. 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.

Program Example

You should understand the maintenance of linked lists as explained in the previous worksheet. The program will thus make more sense. The following program example is a simple line editor though the code is quite long, which allows you to add and delete lines. As in other worksheets and tutorials at Tenouk.com, the program does not do any error checking. Error checking such as using exception handling, is left out at this stage so that the main logic is easier to understand. The following example quite long, so, take your time to study it thoroughly.

```
#include <stdio.h>
// for string related functions...
#include <string.h>
// for memory manipulation related functions...
#include <stdlib.h>
#include <malloc.h>

struct Line
{
   char Str[64];
   struct Line *Next;
};

int GetChoice(struct Line *p);
struct Line * AddLine(struct Line *p);
struct Line * DelLine(struct Line *p);

void main(void)
{
   // address of the first node of the list
   struct Line *Head;
   int x;

   // initially the list is empty
   Head = NULL;
   x = GetChoice(Head);
   for( ; x != 3; )
   {
        if(x == 1)
             // add a line and adjust head if needed
             Head = AddLine(Head);
        else
             // delete one and adjust head if needed
             Head = DelLine(Head);
        x = GetChoice(Head);
   }
}

// initially address of first node...
struct Line *DelLine(struct Line *p)
{
   // address of the first node
   struct Line *pStart,       // address of the first node
        *pLast;               // this follows p as p moves forward
   int i, Number;

   printf("\tWhich line number lol? ");
   // get a line number to delete
   scanf_s("%d", &Number, 1);
   // if deleting first node, adjust  pStart
   if(Number == 1)
        pStart = p ->Next;
   // otherwise, find p, the node to remove
   else
   {
        for(pStart = p, i = 1; i < Number; ++i, p = p -> Next)
             pLast = p;
        pLast -> Next = p -> Next;
   }

   // return p to system/memory
   free(p);
   // return starting address to main()
   return (pStart);
}

int GetChoice(struct Line *p)
{
   int i, Choice;

   printf("\nThis is the start of your file so far...\n");
```

```
    // print out the list
    for(i = 1; p != NULL; ++i, p = p->Next)
        printf("%d: %s\n", i, p->Str);
    printf("...and this is the end ");
    printf("(Enter 1 to Insert, 2 to Delete and 3 to Quit)\n\t");
    scanf_s("%d", &Choice, 1);
    // get a choice and return it to main()
    return Choice;
}

struct Line *AddLine(struct Line *p)
{
    int Number, i;
    struct Line *pLast, *pNew, *pStart;
    char StrNew[64];

    pStart = p;
    // get enough memory for storage
    pNew = (struct Line *) malloc(sizeof(struct Line));
    printf("What number will this line be? :");
    scanf_s("%d", &Number, 1);
    // be careful with the line of text that you enter, do not proceed
    // the allocated memory storage....
    printf("\tEnter your line of text lol!: ");
    // flush out the <return> from last scanf_s()
    getchar();
    // get line to add
    gets_s(StrNew, 64);

    for(i = 1; i < Number; ++i, p = p -> Next)
        // find p and pLast, the point of insertion
        pLast = p;
    pNew -> Next = p;
    strcpy_s(pNew->Str, 64, StrNew);

    // if the new node is the first one...
    if(i == 1)
        // return that as the starting address...
        return (pNew);
    // otherwise, complete the link and...
    else
    {
        // Return the starting address
        pLast->Next = pNew;
        return (pStart);
    }
}
```

Sample output:

```
This is the start of your file so far...
...This is the end (Enter 1 to Insert, 2 to Delete and 3 to Quit)
        1
What number will this line be? :1
        Enter your line of text lol!: This is the first line - Line 2

This is the start of your file so far...
1: This is the first line - Line 2
...This is the end (Enter 1 to Insert, 2 to Delete and 3 to Quit)
        1
What number will this line be? :1
        Enter your line of text lol!: This is the second line of text - Line 1

This is the start of your file so far...
1: This is the second line of text - Line 1
2: This is the first line - Line 2
...This is the end (Enter 1 to Insert, 2 to Delete and 3 to Quit)
        1
What number will this line be? :3
        Enter your line of text lol!: Hohoho...third line of text - Line 3

This is the start of your file so far...
1: This is the second line of text - Line 1
2: This is the first line - Line 2
3: Hohoho...third line of text - Line 3
...This is the end (Enter 1 to Insert, 2 to Delete and 3 to Quit)
        2
        Which line number lol? 1

This is the start of your file so far...
1: This is the first line - Line 2
2: Hohoho...third line of text - Line 3
...This is the end (Enter 1 to Insert, 2 to Delete and 3 to Quit)
        1
What number will this line be? :3
        Enter your line of text lol!: The fourth entered line of text Line 4

This is the start of your file so far...
1: This is the first line - Line 2
2: Hohoho...third line of text - Line 3
3: The fourth entered line of text Line 4
...This is the end (Enter 1 to Insert, 2 to Delete and 3 to Quit)
        3
Press any key to continue . . .
```

The Story

Let us start at the screen or the output of the program to see how the program works. The program first shows your file. Initially, there is nothing in the file. From the options provided, we pick option 1, indicating we want to add a line. We also state that we want to add line number 1. The program doesn't allow the user to type in a wrong line number by mistake (you can try by entering other number!). That kind of logic should exist in a program for security and integrity.
We enter the first line of text of our file, the file with one line is shown and then the three options are shown again. Next, we decide to add line number 1, making the previous line of text the second line. Then line number 3 is added to the end and our file is displayed now with its three lines. Next, we delete line number 2 and add another line. This is just an example of the run. We could add and delete any line to our file.
The program does this by using four functions including main(). In main(), a pointer called Head stores the starting address of the linked list. The function called GetChoice() is called in a loop, until a 3 for Quit is entered by the user of the program. If the user types a "1", the AddLine() function is called and if "2" is typed, then DelLine() is called. For both AddLine() and DelLine(), Head is passed and the new starting address of the list is received in it. After returning from these functions, the starting address of the linked list could be different.

GetChoie()

This function also receives Head from main() and Head is copied into p, a pointer to type struct Line. This function prints the entire file before asking for the choice. p goes through the for loop, each time printing the line number called i and the Str member of the Line. When doing the next iteration of the loop, i or the line number is incremented and p is advanced to the address of the next node in the list. Finally, the choice is obtained from the user and that is returned to main().

AddLine()

This function also receives the beginning address of the list in p. p is assigned to pStart. Using malloc (), a pointer to type struct Line is obtained in pNew. Then the string and the line number for inserting the string is obtained from the user. Number contains the line that this new line will be. In the for loop, i is incremented until it is one less than Number. As i is incremented, p is advanced to point to the address of each of the nodes of the list. Also, pLast comes right behind p, making available the address of the previous node when we find the insertion point in the list. Once we find the i where it is equal to Number, p will point to the node that will follow the new node. This is done using the pNew->Next = p; statement. The string containing the line of data is also copied into pNew.Str. If the node is at the beginning of the list, then pNew is returned as the address of the start of the list. Otherwise, we have the last node point to the new node and return the same starting address of the list as what we received.

DelLine()

If the node to be deleted is the first one in the list, then pStart is made to point to the second node in the list. The first node is freed and returns the new starting address. If the node to be deleted is not the first one, then p and pLast are obtained using the same for loop as the AddLine() function. p is the address of the node to be removed. Before it is removed, its next member, which has the address of the next node, is copied into the next member of pLast. Then the node is freed and the starting address of the list is returned.

Exercises

In pointers worksheet we studied how to store and manipulate addresses of memory locations. In this worksheet we will se how we can use pointers, that is once we have the address of a location, we will access its contents. The **asterisk** (*) in the declaration indicates that the variable is a pointer, but the next time the asterisk is used in the program it is called the indirection operator.

1. In the following exercise i is a variable. Its data type is an integer or a whole number. Show the output and answer the questions.

```
#include <stdio.h>

void main()
{
    int i = 56, *p1;  // Statement 1

    p1 = &i;          // Statement 2
    printf("The address of p1 = %p\nThe value of *p1 = %d\n", p1, *p1);     //
Statement 3
}
```

   a. What is the address of i?
   b. What does p1 print?
   c. What does *p1 print?
   d. Does p1 or *p1 print the value of the location whose address is stored in p1?
   e. Why is p1 printed with a %p format specifier and *p1 printed with a %d format specifier?
   f. In which statement is the indirection operator used?

```
The address of p1 = 0012FF60
The value of *p1 = 56
Press any key to continue . . .
```

   a. 0012FF60 (in hexadecimal, your address may be different).
   b. p1 prints the address of the i variable.
   c. *p1 prints the value stored in i.
   d. p1 prints the value of the location.
   e. To print the address location, we use %p format specifier and to print the integer value, we use %d format specifier as usual.
   f. In Statement 1 and Statement 3, that is *p1. When we print *p1, we are dereferencing p1 and printing not the value of p1 but the value of the location whose address is in p1.

2. An easy way of saying "the value of the location whose address is stored in p1" is saying "the value pointed to by p1". Show the output and answer the questions.

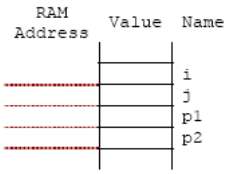
```
#include <stdio.h>

void main()
{
    int i = 56, j, *p1, *p2;     // Statement 1

    p1 = &i;                     // Statement 2
    p2 = p1;                     // Statement 3
    j = *p2;                     // Statement 4
    printf("i = %d, j = %d, *p1 = %d, *p2 = %d\n", i, j, *p1, *p2);
}
```

```
i = 56, j = 56, *p1 = 56, *p2 = 56
Press any key to continue . . .
```

   a. Variable i is initialized to 56.
   b. Variable p1 is initialized to the address of variable i.
   c. Variable p2 is initialized to the address of variable i.
   d. Variable j is initialized to the value stored in p2.
   e. ...

```
i = 56, j = 57, *p1 = 56, *p2 = 56
Press any key to continue . . .
```

   Yes, 1 was added to the address of the value that was pointed to by p2.
   The asterisk had higher priority than addition.
   To see the address of those variable we need to add some code as shown below.



a. In Statement 1, which variable is initialized and to what value? Write it in the diagram.
b. In Statement 2, which variable is initialized and to what value? Write it in the diagram.
c. In Statement 3, which variable is initialized and to what value? Write it in the diagram.
d. In Statement 4, which variable is initialized and to what value? Write it in the diagram.
e. Replace Statement 4 with the following code:

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

And run it. Was 1 added to the address or the value that was pointed to by p2? Which operation had the most priority or precedence, the asterisk or the addition?

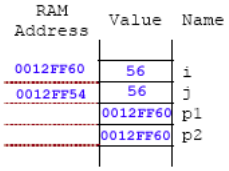
3. Show the output and answer the questions for the following exercise.

```
#include <stdio.h>

void main()
{
    int i = 56, *p1;    // Statement 1

    p1 = &i;            // Statement 2
    *p1 = *p1 + 1;      // Statement 3
    printf("i = %d, *p1 = %d, p1 = %p\n", i, *p1, p1);
}
```

a. p1 has what address in it?
b. p1 points to which variable?
c. p1 points to what value?
d. In Statement 3, the value in which address is changed?
e. In Statement 3, 1 is added to the value in which address?

```
#include <stdio.h>

void main()
{
   int i = 56, j, *p1, *p2; // Statement 1
   p1 = &i;                 // Statement 2
   p2 = p1;                 // Statement 3
   j = *p2;                 // Statement 4
   printf("i = %d, j = %d, *p1 = %d, *p2 = %d\n", i, j, *p1, *p2);
   printf("&i = %p,\n&j = %p,\np1 = %p,\np2 = %p\n", &i, &j, p1, p2);
}
```

```
i = 56, j = 56, *p1 = 56, *p2 = 56
&i = 0012FF60,
&j = 0012FF54,
p1 = 0012FF60,
p2 = 0012FF60
Press any key to continue . . .
```




```
i = 57, *p1 = 57, p1 = 0012FF60
Press any key to continue . . .
```

a. p1 has an address of variable i.
b. p1 points to variable i (the address of variable i).
c. p1 points to the value of variable i's address.
d. The value in address p1 was changed.
e. 1 was added to the value stored at address of variable i.

To see clearly, we add another code after the Statement 2 as shown below. The address of i was not changed but the value stored in i was changed.

```
#include <stdio.h>

void main()
{
   int i = 56, *p1; // Statement 1

   p1 = &i;         // Statement 2
   printf("i = %d, *p1 = %d, p1 = %p\n", i, *p1, p1);
   *p1 = *p1 + 1; // Statement 3
   printf("i = %d, *p1 = %d, p1 = %p\n", i, *p1, p1);
}
```

```
i = 56, *p1 = 56, p1 = 0012FF60
i = 57, *p1 = 57, p1 = 0012FF60
Press any key to continue . . .
```

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

```
#include <stdio.h>

void main()
{
    int i = 56, j = 12, k, *p1 = &i, *p2 = &j, *p3 = &k;

    printf("p1 = %p, p2 = %p, p3 = %p\n", p1, p2, p3);

    *p3 = *p1;      // Statement 1
    *p1 = *p2;      // Statement 2
    *p2 = *p3;      // Statement 3
    printf("p1 = %p, p2 = %p, p3 = %p\n", p1, p2, p3);
    printf("i = %d, j = %d \n", i, j);
}
```

a. p1, p2 and p3 point to which variables?
b. p1 and p2 point to which values?
c. Give the order for these statements:

_____ Whatever p1 points to is assigned the value of whatever p2 points to. The location which p1 points to is changed.
_____ Whatever p3 points to is used to change the location which p2 points to.
_____ The value that p3 points to is changed to the value which p1 points to.

d. Which of the three statements in the exercise are equivalent to each of the statements given below?

_____ j = k;
_____ k = i;
_____ i = j;

```
p1 = 0012FF60, p2 = 0012FF54, p3 = 0012FF48
p1 = 0012FF60, p2 = 0012FF54, p3 = 0012FF48
i = 12, j = 56
Press any key to continue . . .
```

a. p1 points to i variable. p2 points to j variable and p3 points to k variable.
b. p1 points to 56 and p2 points to 12.
c. The order is:

2 (Statement 2) Whatever p1 points to is assigned the value of whatever p2 points to. The location which p1 points to is changed.
3 (Statement 3) Whatever p3 points to is used to change the location which p2 points to.
1 (Statement 1) The value that p3 points to is changed to the value which p1 points to.

d. ...

Statement 2, j = k;
Statement 1, k = i;
Statement 3, i = j;

e. The value of i changed to the value of j and the value of j changed to i. From the previous answer, the value of i was assigned to k (Statement 1) and then the value of k was assigned to j (Statement 2). So the value of i was changed to j. In Statement 3, the value of j was assigned to i. So, the value of j was changed to i.

e. Were the values of i and j changed? Why or why not?
f. Were the values of p1 and p2 changed? Why or why not?

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

```
#include <stdio.h>

void FunStuff(int *, int *, int *);

void main()
{
    int i = 56, j = 12, k = 99, *p1 = &i, *p2 = &j, *p3;

    p3 = &k;

    printf("p1 = %p, p2 = %p, p3 = %p\n", p1, p2, p3);
    FunStuff(&i, p2, p3);
    printf("\ni = %d, j = %d, k = %d\n", i, j, k);
    printf("\np1 = %p, p2 = %p, p3 = %p\n", p1, p2, p3);
}
// the parameter should match with the &i, p2 and p3 in order
// or i, j, k
void FunStuff(int *q, int *r, int *s)
{
    *q = *r + *s;          // Statement 1
    *r = *s;               // Statement 2
    *s = 0;                // Statement 3
    r = r + 1;             // Statement 4
    s = s + 1;             // Statement 5
}
```

main()

Address	Value	Name	Address	Value	Name
		i			p1
		j			p2
		k			p3

FunStuff()

Address	Value	Name
		q
		r
		s

a. In the diagram, show the addresses and initial values of i, j and k. Also, show the values of p1, p2 and p3.
b. In FunStuff() can q, r and s store integers? Can they store the addresses of integers?
c. In the diagram, show the values at which address is changed? This is the same location of which variable in main()?
d. In Statement 1, the value at which address is changed? This is the same location of which variable in main()?
e. In Statement 1, the values from which two hex addresses are added?
f. In Statement 2, the value at which location is changed? This is the same location of which variable in main()?
g. Similarly, the location of which variable in main() is set to zero?
h. In Statement 4 and 5, when s and r were changed, were any of the values in main() changed? Why or why not?

f. The values of p1 and p2 were not changed because these are addresses where the real values were stored. Only the stored values were changed.

```
p1 = 0012FF60, p2 = 0012FF54, p3 = 0012FF48

i = 111, j = 99, k = 0

p1 = 0012FF60, p2 = 0012FF54, p3 = 0012FF48
Press any key to continue . . .
```

a. Filing the diagram:

main()

Address	Value	Name	Address	Value	Name
0012FF60	111	i	0012FF60		p1
0012FF54	99	j	0012FF54		p2
0012FF48	0	k	0012FF48		p3

FunStuff()

Address	Value	Name
0012FF60	111	q
0012FF58	???	r
0012FF4C	???	s

b. Yes, in FunStuff() q, r and s can store integers and addresses of integers.
c. Variable i.
d. The value at the address of i.
e. Values at addresses of j and k.
f. The value at address of r and it is the same location as j in main().
g. The value at address of k variable in main() was set to 0.
h. The values in main() were not changed because Statement 4 and 5 referred to the changing of r and s addresses respectively.

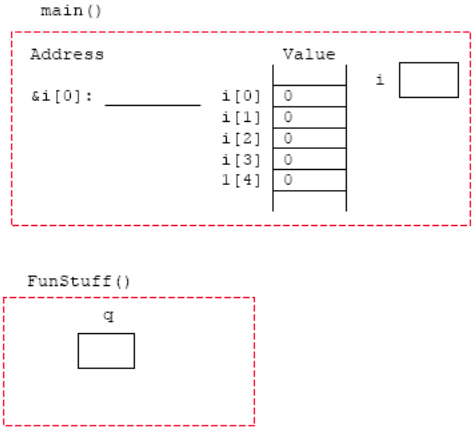
To see more clearly, the following is the previously edited code and study the code. Notice that the values of *r and *s as in the FunStuff() function contain rubbish because the addresses were changed, pointing to the addresses that don't have any value been initialized.

```
#include <stdio.h>

void FunStuff(int *, int *, int *);

void main()
{
   int i = 56, j = 12, k = 99, *p1 = &i, *p2 = &j, *p3;

   p3 = &k;
   printf("In main()\n");
   printf("p1 = %p, p2 = %p, p3 = %p\n", p1, p2, p3);
   FunStuff(&i, p2, p3);
   printf("\nIn main()\n");
   printf("i = %d, j = %d, k = %d\n", i, j, k);
   printf("p1 = %p, p2 = %p, p3 = %p\n", p1, p2, p3);
}

void FunStuff(int *q, int *r, int *s)
{
   printf("\nIn FunStuff()\n");
   *q = *r + *s;     // Statement 1
   *r = *s;          // Statement 2
   *s = 0;           // Statement 3
   r = r + 1;        // Statement 4
   s = s + 1;        // Statement 5
   printf("*q = %d, *r = %d, *s = %d\n", *q, *r, *s);
   printf("q = %p, r = %p, s = %p\n", q, r, s);
}
```

```
In main()
p1 = 0012FF60, p2 = 0012FF54, p3 = 0012FF48

In FunStuff()
*q = 111, *r = -858993460, *s = -858993460
q = 0012FF60, r = 0012FF58, s = 0012FF4C

In main()
i = 111, j = 99, k = 0
p1 = 0012FF60, p2 = 0012FF54, p3 = 0012FF48
Press any key to continue . . .
```

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

```
#include <stdio.h>

void FunStuff(int *);

void main()
{
    int i[5] = {0, 0, 0, 0, 0};

    printf("&i[0] = %p, i = %p\n", &i[0], i);
    FunStuff(i);
    printf("\ni[0] = %d, i[1] = %d, i[2] = %d, i[3] = %d\n", i[0], i[1], i[2], i[3]);
}

void FunStuff(int *q)                  // Statement 1
{
    *q = 10;                           // Statement 2
    ++q;                               // Statement 3
    *q = 20;                           // Statement 4
    *(q + 1) = 30;                     // Statement 5
    printf("\nq = %p\n", q);           // Statement 6
    q[2] = 40;                         // Statement 7
}
```



a. Show the address of i[0] and the values of i and q in the diagram.
b. In Statement 2, what was the value of q? Which slot of i[ ] was changed here?
c. In Statement 4, what was the value of q? Which slot of i[ ] was changed here?
d. In Statement 7, what was the value of q? Which slot of i[ ] was changed here?
e. Change Statement 1 (and the prototype) as follows. Is there any difference in the execution?

```
void FunStuff(int q[ ])
```

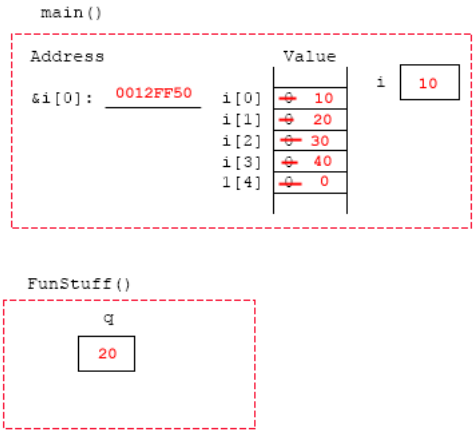
f. If q is a pointer, as it was initially, is the following statement true?

```
*(q + 3) == q[3]?
```

g. If q is an array, as it was in question e, is the following true?

```
*(q + 3) == q[3]?
```

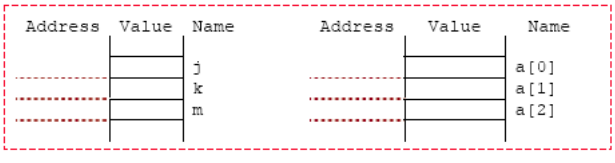
```
&i[0] = 0012FF50, i = 0012FF50

q = 0012FF54

i[0] = 10, i[1] = 20, i[2] = 30, i[3] = 40
Press any key to continue . . .
```



a.
b. The current value of q is 10. The i[0] was changed.
c. The current value of q is 20. The i[1] was changed. The ++q increment the address by 1. So q now pointed to the second slot of the array.
d. The current value of q is 40. The i[3] was changed.
e. No there is no difference in the execution and the output. In this case we pass to FunStuff() the first address (q[]) of the array and previously we pass a pointer (*q) to the first element of the array.
f. The statement is true. Both side refer to the value stored and in this case it refer to the value of fourth element, that is 0.
g. Yes it is true, same as in f.

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

```
#include <stdio.h>

void main(void)
{
  int i, j = 1, k = 2, m = 3, *a[3];

  a[0] = &j;
  a[1] = &k;
  a[2] = &m;
  for(i = 0; i <= 2; ++i)
      printf("*a[%d] = %d, a[%d] = %p\n", i, *a[i], i, a[i]);
}
```



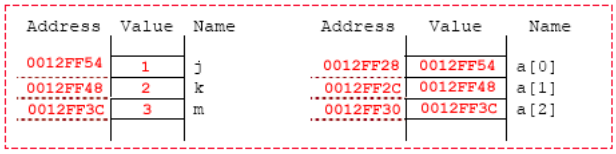
a. Fill in all the values you can in the diagram.
b. If the asterisk weren't used when defining a, then a[ ] would be an array of what?
c. Because there is an asterisk when defining a[ ], a[ ] is an array of pointers to what?
d. Why is a %p format specifier used when printing a[i]?
e. Why is a %d format specifier used when printing *a[i]?

```
*a[0] = 1, a[0] = 0012FF54
*a[1] = 2, a[1] = 0012FF48
*a[2] = 3, a[2] = 0012FF3C
Press any key to continue .
```

a. Let add some more code to the printf() to see the address of the array.

```
printf("*a[%d] = %d, a[%d] = %p, &a[i] = %p\n", i,*a[i],i,a[i],&a[i]);
```

```
*a[0] = 1, a[0] = 0012FF54, &a[i] = 0012FF28
*a[1] = 2, a[1] = 0012FF48, &a[i] = 0012FF2C
*a[2] = 3, a[2] = 0012FF3C, &a[i] = 0012FF30
Press any key to continue . . .
```

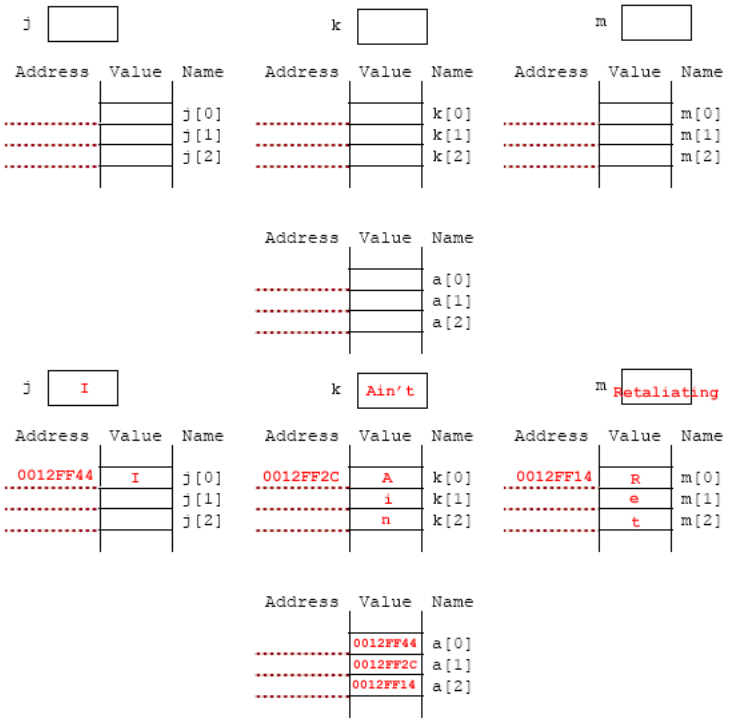


b. a[ ] will be an array of integers.
c. a[ ] is an array of pointers to integers.
d. %p was used to print an address for the a[i].
e. %p was used to print an integer value for the *a[i].

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

```
#include <stdio.h>

void main(void)
{
   int i;

   char j[15] = "I",
        k[15] = "Ain't",
       m[15] = "Retaliating", *a[3];

   a[0] = j;
   a[1] = k;
   a[2] = m;
   printf("j = %s, j = %p, &j[0] = %s, &j[0] = %p\n\n", j, j, &j[0], &j[0]);
   for(i = 0; i <= 2; ++i)
        printf("a[i] = %s, a[i] =  %p\n", a[i], a[i]);
}
```

```
j = I, j = 0012FF44, &j[0] = I, &j[0] = 0012FF44

a[0] = I, a[0] =  0012FF44
a[1] = Ain't, a[1] =  0012FF2C
a[2] = Retaliating, a[2] =  0012FF14
Press any key to continue . . .
```



a. Fill in all the values you can in the diagram.
b. a[ ] is an array of pointers to what type of data?
c. Which of these conditions is true?

   j == &j[0]?
   j == a[0]?

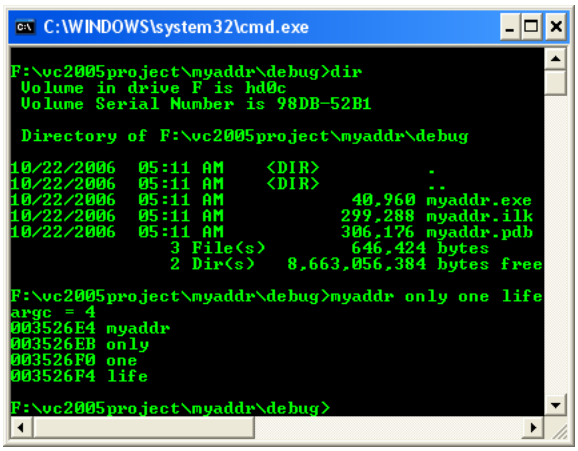
d. The array j[ ] holds characters or addresses?
e. The array a[ ] holds characters or addresses?
f. When printing j or a[0] using a %s format, what is printed?
g. When printing j or a[0] using a %p format, what is printed?
h. When printing the address of a character string, what format specifier is used to print its hex address? The string characters?

a. See the above Figure.
b. String data type.
c. Both are true based on the output sample. Both side of the equality operator are addresses.
d. The array j[ ] holds characters.
e. The array a[ ] hold addresses.
f. A string stored there would be printed.
g. The address of the first array element would be printed.
h. %s was used to print the string characters while %p was used to print the address of a character string.

9. When you run the following program, first make it only executable. Get to the prompt of your operating system and type "myaddr only one life", where "myaddr" is the name of your executable file (your project file name). Change accordingly if your executable file name is different. Show the output and answer the questions.

```
#include <stdio.h>

// main() function variation
int main(int argc, char *argv[ ])
{
   printf("argc = %d\n", argc);
   printf("%p %s \n", argv[0], argv[0]);
   printf("%p %s \n", argv[1], argv[1]);
   printf("%p %s \n", argv[2], argv[2]);
   printf("%p %s \n", argv[3], argv[3]);
   // return 0 means back to the system/OS
   return 0;
}
```

Find the executable, run it at command prompt as shown on the right instead of running it through your compiler as usually done. Your project files and the executable is under the **debug** directory as shown.



a. See the following Figure.

Address	Value	Name	Address	Value	Name
	"myaddr"				argc
	"only"				argv[0]
	"one"				argv[1]
	"life"				argv[2]

Address	Value	Name	Address	Value	Name
003526E4	"myaddr"			4	argc
003526EB	"only"			003526E4	argv[0]
003526F0	"one"			003526EB	argv[1]
003526F4	"life"			003526F0	argv[2]
				003526F4	argv[3]

a. Fill in all the values you can in the diagram.
b. Remove the last printf() statement from this exercise. After recompiling it, run it the following way.

"myaddr soon past?"

Change the executable name accordingly to yours and show the output.

c. What is the data type of argc and what does it represent?
d. What is in argv[0] and to what value does it point?
e. What is in argv[1] and to what value does it point?
f. Does main() receive any arguments? How many?
g. Does main() return any values to the operating system?

b. See the following Figure.

```
D:\extraproject\myaddr\debug>myaddr soon past?
argc = 3
00352630 myaddr
00352637 soon
0035263C past?
00000000 (null)

D:\extraproject\myaddr\debug>
```

c. argc data type is an integer (int). argc stands for argument counter that represent the number of the command line arguments including the executable name (that stored at argv[0]).
d. argv[0] is the first array of string and it points to the executable name, the first command line argument. argv stands for argument vector that represent command line arguments.
e. argv[1] is the second array of string and it points to the second string of the command line arguments.
f. Yes, main() receives three arguments as in (b) and four arguments as in (a).
g. Yes, main() returns an integer 0 that represents no error.

Note: A complete story and working program examples of the command line arguments for C/C++ programs can be found in Command Line Arguments.