**Assignment 2**

Ans 1) It is true that there are 4 arithmetic operations in pointers i.e. ++, --, +, and –.

Code:

#include <stdio.h>

int main()

{

int i=10;

int \*j=&i;

printf("\n value %d",i++);

printf("\n value %d",\*j++);

printf("\n value %d",i--);

printf("\n value %d",(\*j)--);

printf("\n address %p",j++);

printf("\n adderss %p",j--);

printf("\n address %p",j+1);

printf("\n address %p",j-1);

return 0;

}

Output:

 value 10

 value 11

 value 11

 value -62410184

 address 0x7ffffc47b237

 adderss 0x7ffffc47b23b

 address 0x7ffffc47b23b

 address 0x7ffffc47b233

**...Program finished with exit code 0**

**Press ENTER to exit console.**

Note:

We cannot use arithmetic operations on two pointers since pointers are storing memory addresses of the value and we cannot perform arithmetic operations on pointers

For example:

int main()

{

int i=10;

int j=11;

int \*m=&i;

int \*n=&j;

printf("\n %p",m+n);

return 0;

}

On compilation this will show an error in the code.

main.c: In function ‘main’:

main.c:33:21: error: invalid operands to binary + (have ‘int \*’ and ‘int \*’)

printf("\n %p",m+n);

^

Ans 2) It is true that we can create the array to store the address of the pointers that is by creating array of pointers.

Code:

#include <stdio.h>

int main()

{

int i=10;

int j=11;

int k=12;

int l=13;

int \*a[4];

a[0]=&i;

a[1]=&j;

a[2]=&k;

a[3]=&l;

printf("\n %p",&i);

printf("\n %p",a[0]);

printf("\n %p",&j);

printf("\n %p",a[1]);

printf("\n %p",&k);

printf("\n %p",a[2]);

printf("\n %p",&l);

printf("\n %p",a[3]);

return 0;

}

Output:

0x7ffe4f113690

 0x7ffe4f113690

 0x7ffe4f113694

 0x7ffe4f113694

 0x7ffe4f113698

 0x7ffe4f113698

 0x7ffe4f11369c

 0x7ffe4f11369c

**...Program finished with exit code 0**

**Press ENTER to exit console.**

Ans 3) true, we can assign pointer to a pointer which is a double pointer.

Code:

#include <stdio.h>

int main()

{

int i=10;

int \*j=&i;

int \*\*k=&j;

printf("\n value of i %d",i);

printf("\n value of i %d",\*j);

printf("\n value of i %d",\*\*k);

printf("\n address of i %p",&i);

printf("\n address of i %p",j);

printf("\n address of i %p",\*k);

printf("\n address of j %p",&j);

printf("\n address of j %p",k);

printf("\n address of k %p",&k);

return 0;

}

Output:

value of i 10

 value of i 10

 value of i 10

 address of i 0x7ffe9f3ad10c

 address of i 0x7ffe9f3ad10c

 address of i 0x7ffe9f3ad10c

 address of j 0x7ffe9f3ad110

 address of j 0x7ffe9f3ad110

 address of k 0x7ffe9f3ad118

**...Program finished with exit code 0**

**Press ENTER to exit console.**

Ans 4) True, it will if the same result if we pass the reference or the address in the function.

Code :

#include <stdio.h>

void swap2(int \*a,int \*b);

void swap2(int \*a,int \*b);

int main()

{

int i=10;

int j=20;

int \*p=&i;

int \*q=&j;

swap2(p,q);

int k=10;

int l=20;

printf("\n %d %d",i,j);

swap3(&k,&l);

printf("\n %d %d",k,l);

return 0;

}

void swap2(int \*a,int \*b)

{

int temp;

temp=\*a;

\*a=\*b;

\*b=temp;

}

void swap3(int \*a,int \*b)

{

int temp;

temp=\*a;

\*a=\*b;

\*b=temp;

}

Output:

**main.c:22:5: warning: implicit declaration of function ‘swap3’ [-Wimplicit-function-declaration]**

**main.c:33:6: warning: conflicting types for ‘swap3’**

**main.c:22:5: note: previous implicit declaration of ‘swap3’ was here**

 20 10

 20 10

**...Program finished with exit code 0**

**Press ENTER to exit console.**

Ans 5) True, we can return a pointer in the function.

Code:

#include <stdio.h>

int f(int \*k);

int main()

{

int i=10;

int \*j=&i;

int \*\*c=f(j);

printf("\n %p",c);

return 0;

}

int f(int \*k)

{

int \*\*c=&k;

return \*c;

}

Output:

**main.c:15:13: warning: initialization makes pointer from integer without a cast [-Wint-conversion]**

**main.c:22:12: warning: return makes integer from pointer without a cast [-Wint-conversion]**

 0xffffffffdb0a08ec

**...Program finished with exit code 0**

**Press ENTER to exit console.**

Note: This will give correct result in case of double pointer otherwise it will give you segmentation fault or garbage address of a pointer or errors in the code and its execution.