INDEX

Name : Nisarg .K. Amlani
Roll : Ce001
Id : 22ceueg082

LAB : 01

---

1. Find minimum of two integer numbers in an assembly function (FAR Procedure). The numbers are passed from main program in C and result is passed back from assembly function to the C program.

Main.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>

int n1 = 20 , n2 = 30;
int extern far fmin(int , int );

void  main()
{
    clrscr();
    printf("sum = %d",fmin(n1,n2));
    getch();
}
```

Fmin.asm

```
public _fmin
extrn _n1:word , _n2:word
_code segment
```

```
assume cs:_code

_fmin proc far

mov ax , _n1
cmp ax , _n2
jg _grt
mov ax , _n2
_grt: mov ax , _n1
ret
_fmin endp
int 03h
_code ends
end
```

Output

```
min = 20
```

2. Convert temperature given in Celsius to Fahrenheit using assembly function (FAR Procedure). The assembly function is called from main program in C.

Ctf.asm

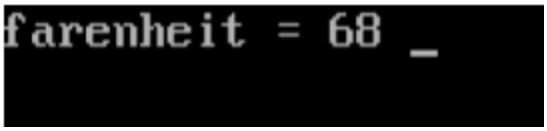
```
public _ctof
extrn _c:word

_code segment
```

```
assume cs:_code
_ctof proc far

mov ax,_c
mov bx , 9
mul  bx
mov bx , 5
div bx
add ax , 32
ret

_ctof endp
int 03h
_code ends
end
```

Main.c

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>

int c = 20;
int extern far ctof(int  );

void  main()
{
    clrscr();
    printf("farenheit = %d",ctof(c));
    getch();
}
```

Output



3. Calculate the LCM of two integer numbers in an assembly function (FAR Procedure). The numbers are passed from main program in C and LCM is passed back from the assembly function back to C program.

lcm.asm

```
public _lcm

extrn _num1:word, _num2:word

_code segment
assume cs:_code

_lcm proc far
   mov ax, _num1
   mov bx, _num2

loop1:
   mov dx, ax
   cmp ax, bx
   jnc swap
   mov ax, bx
   sub ax, bx

swap:
   mov ax, bx
   mov bx, dx
   jnz loop1

   ret
_lcm endp
```

```
int 3
_code ends
end
```

Main.c

```
#include<stdio.h>
#include<conio.h>

int num1=5, num2=3, ans;
int extern far lcm(int, int);

void main()
{
   clrscr();
   ans = lcm(num1, num2);
   printf("lcm is %d", ans);
   getch();
}
```

```
lcm is 1_
```

Name : Nisarg Amlani
Roll : CE001
ID : 22ceueg082

LAB : 2

Q1) Write an ALP to do addition of three floating point numbers using 8087 instruction set.

Test case data: x = 3.5 y = 5.0 z = 2.2

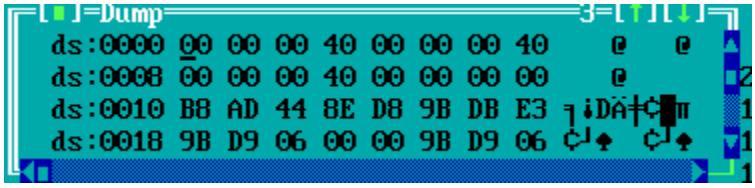
Code)

```
data segment
a dd 2.0
b dd 2.0
c dd 2.0
d dd ?
data ends

code segment

assume cs:code , ds:data
start:
mov ax, data
mov ds, ax
finit
fld a
fld b
fld c
fadd
fadd
fst d
int 03h
code ends
end start
```


Valid ST(0) 2
Valid ST(1) 2
Valid ST(2) 2
Empty ST(3)
Empty ST(4)
Empty ST(5)
Empty ST(6)
Empty ST(7)
im=1 ie=0
dm=1 de=0
zm=1 ze=0
om=1 oe=0
um=1 ue=0
pm=1 pe=0
iem=0 ir=0
pc=3 cc=1
rc=0 st=5
ic=0


80486 IPTR=00000 OPCODE=000 OPTR=00000 2
Valid ST(0) 4
Valid ST(1) 2
Empty ST(2)
Empty ST(3)
Empty ST(4)
Empty ST(5)
Empty ST(6)
Empty ST(7)
im=1 ie=0
dm=1 de=0
zm=1 ze=0
om=1 oe=0
um=1 ue=0
pm=1 pe=0
iem=0 ir=0
pc=3 cc=1
rc=0 st=6
ic=0

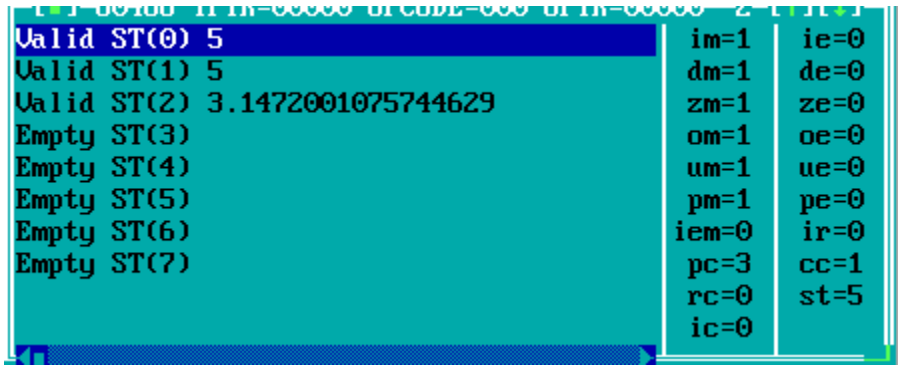

Valid ST(0) 6
Empty ST(1)
Empty ST(2)
Empty ST(3)
Empty ST(4)
Empty ST(5)
Empty ST(6)
Empty ST(7)
im=1 ie=0
dm=1 de=0
zm=1 ze=0
om=1 oe=0
um=1 ue=0
pm=1 pe=0
iem=0 ir=0
pc=3 cc=1
rc=0 st=7
ic=0

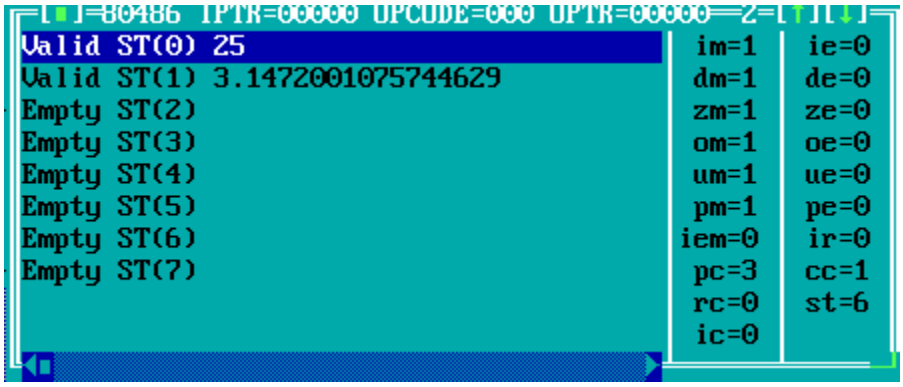

Dump
ds:0000 00 00 00 40 00 00 00 40 @ @
ds:0008 00 00 00 40 00 00 00 00 @
ds:0010 B8 AD 44 8E D8 9B DB E3
ds:0018 9B D9 06 00 00 9B D9 06

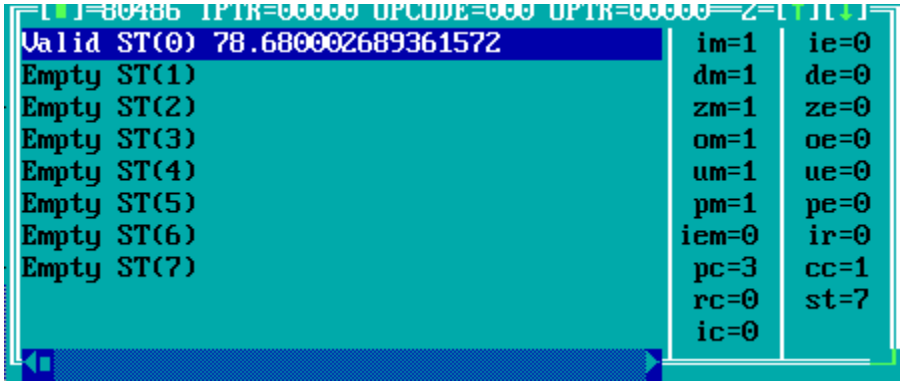
Q2) Write an ALP to find area of a circle using 8087 instruction set.
Test case data: Pi = 3.1472 radius = 5.0

Code)

```
data segment
pi dd 3.1472
r dd 5.0
res dd ?
data ends

code segment

assume cs:code , ds:data
start:
mov ax, data
mov ds, ax
finit
fld pi
fld r
fld r
fmul
fmul
fst res
int 03h
code ends
end start
```





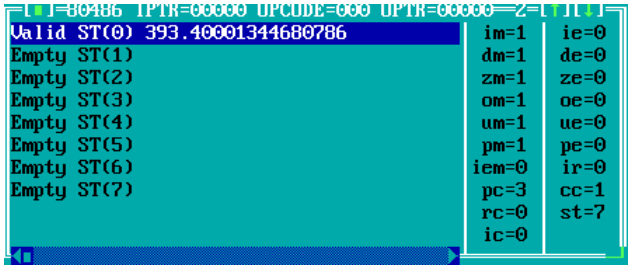


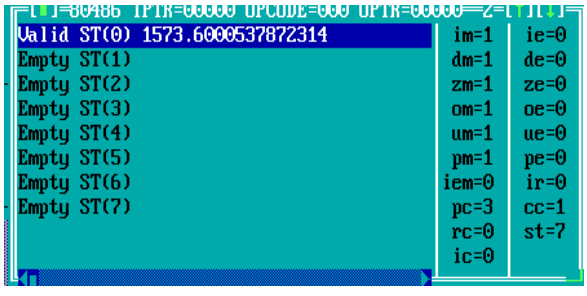
Q3) Write an ALP to find the volume of sphere using 8087 instruction set.
Test case data: Pi = 3.1472 radius = 5.0

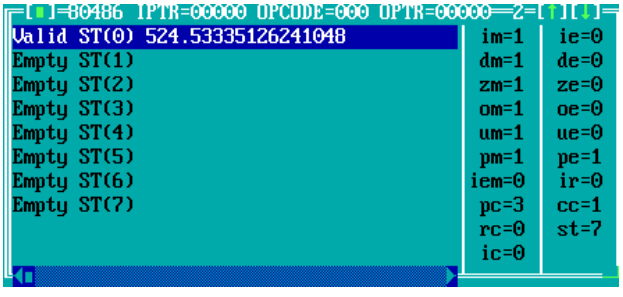
Code)

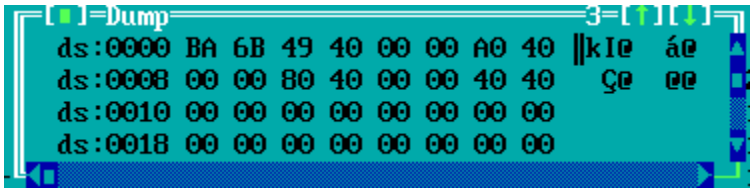
```
data segment
pi dd 3.1472
r dd 5.0
c dd 4.0
c2 dd 3.0
res dd ?
data ends

code segment
```

```
assume cs:code , ds:data
start:
mov ax, data
mov ds, ax
finit
fld r
fmul r
fmul r
fmul pi
fmul c
fdiv c2
fst res
int 03h
code ends
end start
```





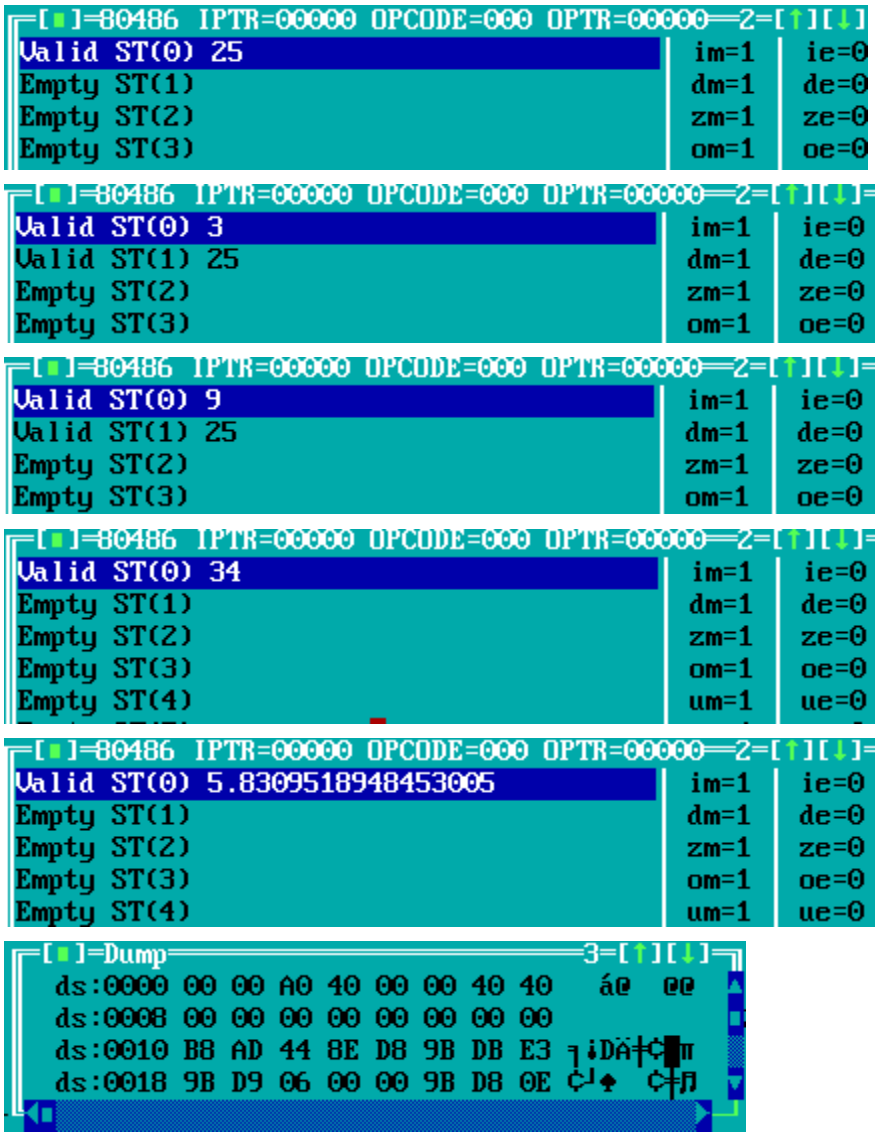




4. Write an ALP to find c = sqrt(a^2 + b^2) .
Test case data: a = 5.0 b = 3.0

Code)

```
data segment
a dd 5.0
b dd 3.0
res dd ?
data ends

code segment

assume cs:code , ds:data
start:
mov ax, data
mov ds, ax
finit
fld a
fmul a
fld b
fmul b
fadd
fsqrt
fst res
int 03h
code ends
end start
```


[■]=80486 IPTR=00000 OPCODE=000 OPTR=00000 2=[↑][↓]
Valid ST(0) 25
Empty ST(1)
Empty ST(2)
Empty ST(3)
im=1 ie=0
dm=1 de=0
zm=1 ze=0
om=1 oe=0
[■]=80486 IPTR=00000 OPCODE=000 OPTR=00000 2=[↑][↓]
Valid ST(0) 3
Valid ST(1) 25
Empty ST(2)
Empty ST(3)
im=1 ie=0
dm=1 de=0
zm=1 ze=0
om=1 oe=0
[■]=80486 IPTR=00000 OPCODE=000 OPTR=00000 2=[↑][↓]
Valid ST(0) 9
Valid ST(1) 25
Empty ST(2)
Empty ST(3)
im=1 ie=0
dm=1 de=0
zm=1 ze=0
om=1 oe=0
[■]=80486 IPTR=00000 OPCODE=000 OPTR=00000 2=[↑][↓]
Valid ST(0) 34
Empty ST(1)
Empty ST(2)
Empty ST(3)
Empty ST(4)
im=1 ie=0
dm=1 de=0
zm=1 ze=0
om=1 oe=0
um=1 ue=0
[■]=80486 IPTR=00000 OPCODE=000 OPTR=00000 2=[↑][↓]
Valid ST(0) 5.8309518948453005
Empty ST(1)
Empty ST(2)
Empty ST(3)
Empty ST(4)
im=1 ie=0
dm=1 de=0
zm=1 ze=0
om=1 oe=0
um=1 ue=0
[■]=Dump 3=[↑][↓]
ds:0000 00 00 A0 40 00 00 40 40 á@ @@
ds:0008 00 00 00 00 00 00 00 00
ds:0010 B8 AD 44 8E D8 9B DB E3 ╕¡DÄ╪¢█π
ds:0018 9B D9 06 00 00 9B D8 0E ¢┘♠ ¢╪♫

i). Convert 89.625 to IEEE single precision.

89.625 → 1011001.101

Sign bit → 0

1.011001101×2^6

S	EXP.	M.
0	10000101	01100110100...0

Hex :- 42B34000H

ii) Convert 89.625 to IEEE double precision format

89.625 → 1011001.101

Sign bit → 0

1.011001101×2^6

Exponent = 6 + 1023 = 1029 → 10000000101

~~1011001101~~

S	E	M
0	10000000101	01100110100...0

Hex :- 4056680000000000H

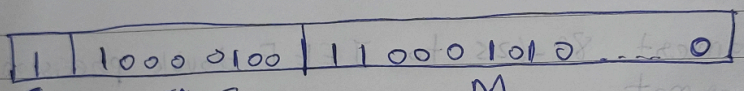
iii) Convert -56.625 to IEEE 754 single precision

56.625 → 00111000.101

Sign bit → 1

Normalized :- 1.11000101×2^5

Exponent = 5 + 127
= 1000 0100



Hex:- C2628000H

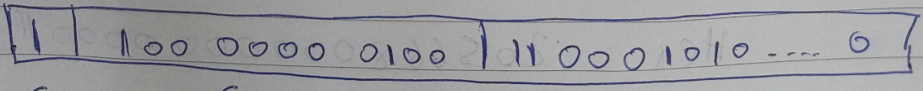
iv) Convert -56.625 to IEEE double precision

-56.625 → 00111000.101

Sign bit → 1

Normalized :- 1.11000101×2^5

Exponent = 5 + 1023 = 1028
100 0000 0100



Hex:- C04CA0000000000H

Name :- Nisarg Amlani
Roll :- ce001
ID :- 22ceueg082

LAB : 03

---

1. Compute tan(x)

=> x = 45

Code:

```
data segment
th dd 45.0
d dd 180.0
pi dd 3.1428
res dd ?
data ends

code segment

assume cs:code , ds:data
start:
mov ax, data
mov ds, ax
finit
fld th
fldpi
fmul
fdiv d
fptan
fxch
fst res
int 03h
code ends
end start
```

Output :

2. Compute sin(x)

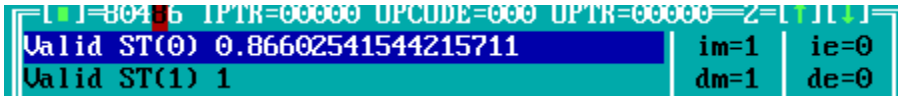
=> x = 60

Code :

```
data segment
th dd 60.0
d dd 180.0
one dd 1.0
res dd ?
data ends

code segment

assume cs:code , ds:data
start:
mov ax, data
mov ds, ax
finit
fld th
fldpi
fmul
fdiv d
fptan
fxch
fst res
fld one
fld res
fmul res
fadd
fsqrt
fdiv
fst res
int 03h
```

```
code ends
end start
```

Output :



3. Compute cos(x)

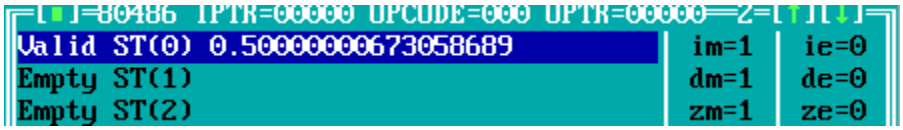
=> x = 60

Code :

```
data segment
th dd 60.0
d dd 180.0
one dd 1.0
res dd ?
data ends

code segment

assume cs:code , ds:data
start:
mov ax, data
mov ds, ax
finit
fld th
fldpi
fmul
fdiv d
fptan
fxch
fstp res
fld one
fld res
fmul res
fadd
fsqrt
fdiv
```

```
fst res
int 03h
code ends
end start
```

Output:



5. Compute xsqrt(y) + ysqrt(x)

=> x= 4.0 , y = 4.0

Code:

```
data segment
x dd 4.0
y dd 4.0
res dd ?
data ends

code segment

assume cs:code , ds:data
start:
mov ax, data
mov ds, ax
finit
fld x
fld y
fsqrt
fmul
fld y
fld x
fsqrt
fmul
fadd
fst res

int 03h
code ends
```

```
end start
```

Output :

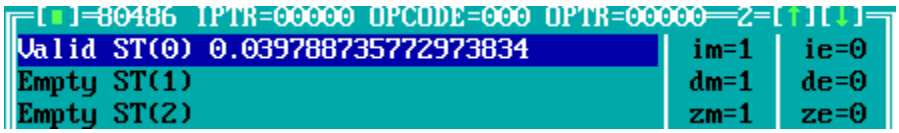
6. Find resonance frequency

=> l = 4.0 , c = 4.0

Code : .

```
data segment
c1 dd 2.0
c2 dd 1.0
l dd 4.0
c dd 4.0
res dd ?
data ends

code segment

assume cs:code , ds:data
start:
mov ax, data
mov ds, ax
finit
fld c2
fld l
fmul c
fsqrt
fmul c1
fldpi
fmul
fdiv
fst res
int 03h
code ends
end start
```

Output:



7. Compute tan inverse (y/x)

=> x = 1.0 y = 1.0

Code :

```
data segment
x dd 1.0
y dd 1.0
c1 dd 180.0
res dd ?
data ends

code segment

assume cs:code , ds:data
start:
mov ax, data
mov ds, ax
finit
fld y
fld x
fpatan
fmul c1
fldpi
fdiv
fst res
int 03h
code ends
end start
```

Output:

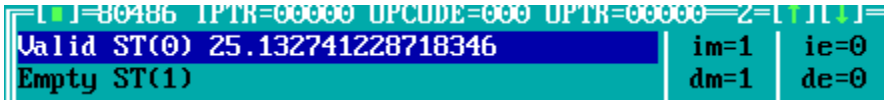
8. Compute area of a cone.

=> l = 2.0, r = 2.0

Code :

```
data segment
l dd 2.0
r dd 2.0
c1 dd 180.0
res dd ?
data ends

code segment

assume cs:code , ds:data
start:
mov ax, data
mov ds, ax
finit
fldpi
fld l
fld r
fadd
fmul
fmul r
fst res
int 03h
code ends
end start
```

Output:



Name : Nisarg Amlani .k.
Roll : CE001
ID : 22ceueg082

LAB : 4

AIM: 8087 programs using 8086 and 8087 instruction set

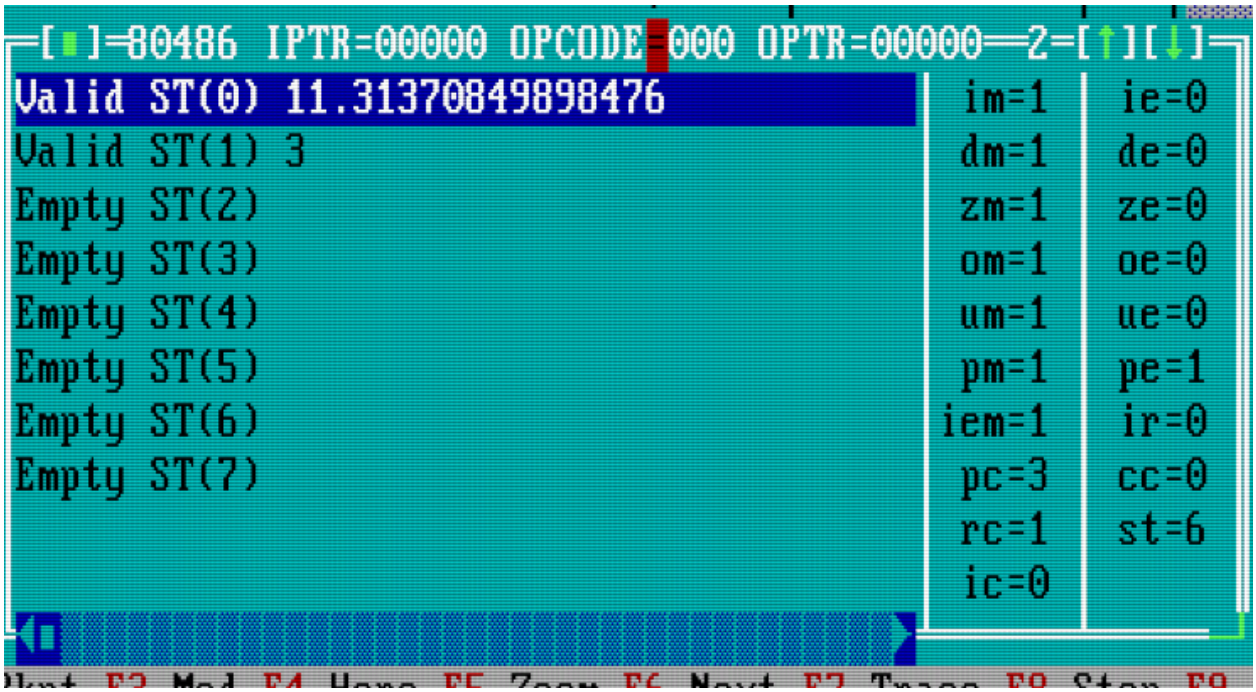
1. Compute x^y. x and y can be integer or real.

=> x = 2.0 , y = 3.5

Code:

```
data segment
x dd 2.0
y dd 3.5
power dd ?
intpower dd ?
res dd ?
cw dw 07ffh
data ends

code segment
assume cs:code, ds:data
start:
   mov ax, data
   mov ds, ax
   finit

   fld y          ; Load exponent (y) onto the FPU stack
   fld x          ; Load base (x) onto the FPU stack
   fyl2x          ; Calculate y * log2(x), result in st(0)
   fst power      ; Store the calculated power (y * log2(x))
   fldcw cw       ; Load control word for truncation
   frndint        ; Round st(0) to the nearest integer
   fst intpower   ; Store the integer part of the power

   fld power      ; Reload the power value
   fxch           ; Exchange st(0) and st(1) (intpower and fractional part)
```

```
    fsub            ; Subtract intpower from power to isolate the fractional part
    f2xm1           ; Calculate 2^(fractional part) - 1
    fld1            ; Load constant 1 onto the FPU stack
    fadd            ; Add 1 to calculate 2^(fractional part)

    fld intpower    ; Reload the integer part of the power
    fxch            ; Exchange st(0) and st(1) (fractional and integer parts)
    fscale          ; Scale result by 2^intpower (final result in st(0))
    fst res         ; Store the final result in memory

    int 3
code ends
end start
```

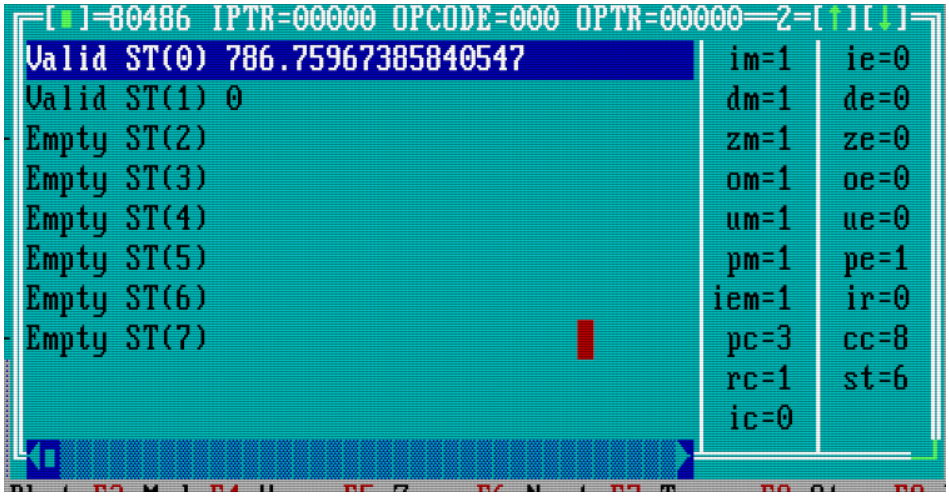


2. Compute P × (1 +r/ 100)^n. Example: P = 500.0, r =12% and n =4.0

Code:

```
data segment
r dd 12.0
n dd 4.0
```

```
p dd 500.0
divider dd 100.0
power dd ?
 intpower dd ?
res dd ?
cw dw 07ffh
data ends

code segment
assume cs:code, ds:data
start:
   mov ax, data
   mov ds, ax
   finit

   fld n             ; Load n (exponent) onto the FPU stack
   fld r             ; Load r (interest rate) onto the FPU stack
   fld divider       ; Load the divider value (100.0) onto the FPU stack
   fdiv              ; Divide st(1) by st(0) (r / divider), result in st(1)
   fld1              ; Load constant 1.0 onto the FPU stack
   fadd              ; Add st(0) and st(1) (r/divider + 1), result in st(0)
   fyl2x             ; Compute n * log2(st(0)) (exponentiation via logarithm), result in st(0)
   fst power         ; Store the calculated power (n * log2(1 + r/100))

   fldcw cw          ; Load control word to configure rounding mode
   frndint           ; Round st(0) to the nearest integer
   fst intpower      ; Store the integer part of the power in memory

   fld power         ; Reload the full power value (n * log2(1 + r/100))
   fxch              ; Exchange st(0) and st(1) (swap full power and integer power)
   fsub              ; Subtract integer part from full power to isolate the fractional part
   f2xm1             ; Compute 2^(fractional part) - 1, result in st(0)
   fld1              ; Load constant 1.0 onto the FPU stack
   fadd              ; Add 1 to compute 2^(fractional part)

   fld intpower      ; Reload the integer part of the power
   fxch              ; Exchange st(0) and st(1) (swap fractional part result and integer part)
   fscale            ; Scale st(0) by 2^st(1) (compute full 2^power)
   fmul p            ; Multiply the result by principal amount (p)
   fst res           ; Store the final result (res) in memory

   int 3
code ends
end start
```



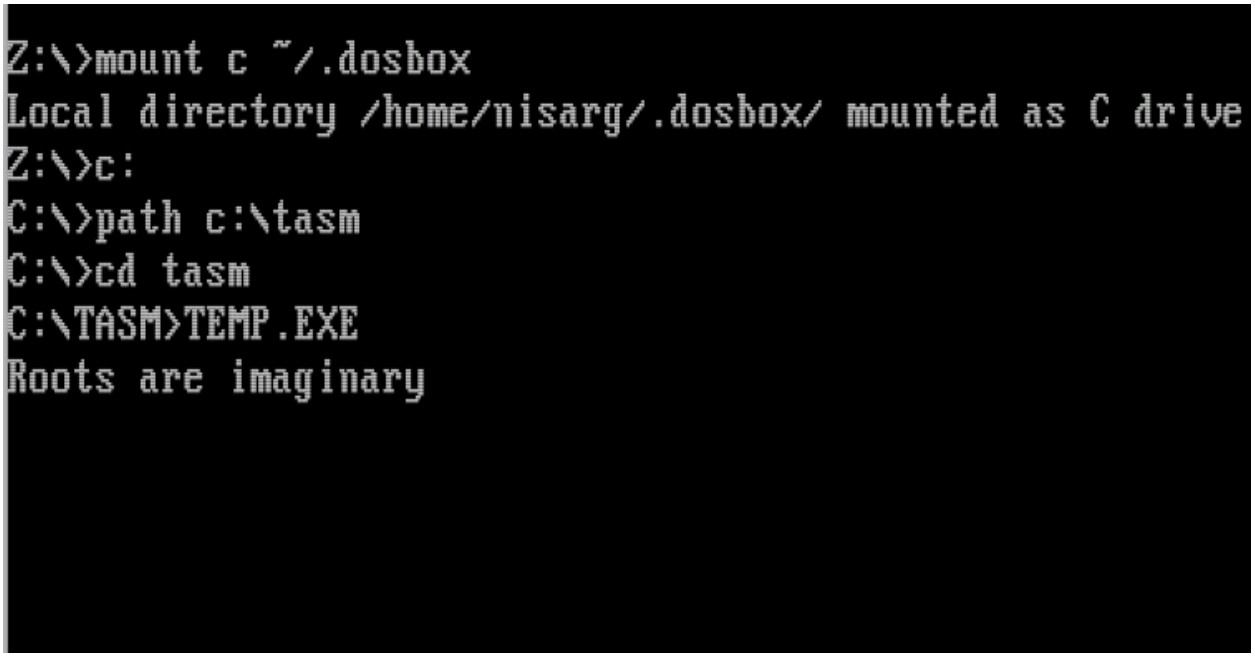
3. Compute roots of quadratic equation. If roots are imaginary display message “Roots are imaginary”.
Example: x^2 + 7x + 10 = 0

Code:

```
data segment
        a dd 1.0
        b dd 7.0
        c dd 15.0
        val dd 4.0
        val1 dd 2.0
        temp dw ?
        alpha dd ?
        ans dd ?
        ans1 dd ?
        str1 db "Roots are imaginary$" ; Message for imaginary roots
data ends

code segment
        Assume cs:code, ds:data

start:
        mov ax, data
        mov ds, ax
```

```
        finit

        ; Calculate the discriminant: b^2 - 4*a*c
        fld b               ; Load coefficient b onto the FPU stack
        fmul b              ; Compute b^2
        fld a               ; Load coefficient a onto the FPU stack
        fmul c              ; Compute a * c
        fmul val            ; Compute 4 * a * c
        fsub                ; Subtract 4 * a * c from b^2
        fsqrt               ; Compute the square root of the discriminant
        fstsw temp          ; Store the FPU status word in temp
        mov ax, temp        ; Move the status word to AX
        and ax, 0001h       ; Check for invalid operation (e.g., sqrt of a negative number)
        cmp ax, 0001h       ; Compare the result with 1 (indicating imaginary roots)
        jz imaginary        ; If discriminant is negative, jump to the "imaginary" label
        fstp alpha          ; Store the square root of the discriminant in alpha

        ; Calculate the first root: (-b + sqrt(discriminant)) / (2 * a)
        fld b               ; Load coefficient b onto the FPU stack
        fchs                ; Negate b (-b)
        fadd alpha          ; Add the square root of the discriminant
        fld a               ; Load coefficient a onto the FPU stack
        fmul val1           ; Multiply a by 2
        fdiv                ; Divide (-b + sqrt(discriminant)) by (2 * a)
        fstp ans            ; Store the result in ans (first root)

        ; Calculate the second root: (-b - sqrt(discriminant)) / (2 * a)
        fld b               ; Load coefficient b onto the FPU stack
        fchs                ; Negate b (-b)
        fsub alpha          ; Subtract the square root of the discriminant
        fld a               ; Load coefficient a onto the FPU stack
        fmul val1           ; Multiply a by 2
        fdiv                ; Divide (-b - sqrt(discriminant)) by (2 * a)
        fstp ans1           ; Store the result in ans1 (second root)

        jmp exit            ; Jump to exit

imaginary:
        ; Print the message "Roots are imaginary"
        mov ah, 09h         ; DOS interrupt for string output
```

```
        lea dx, str1        ; Load the address of the message into DX
        int 21h             ; Execute the interrupt
        jmp exit            ; Jump to exit

exit:
        int 3               ; Terminate the program
code ends
end start
```



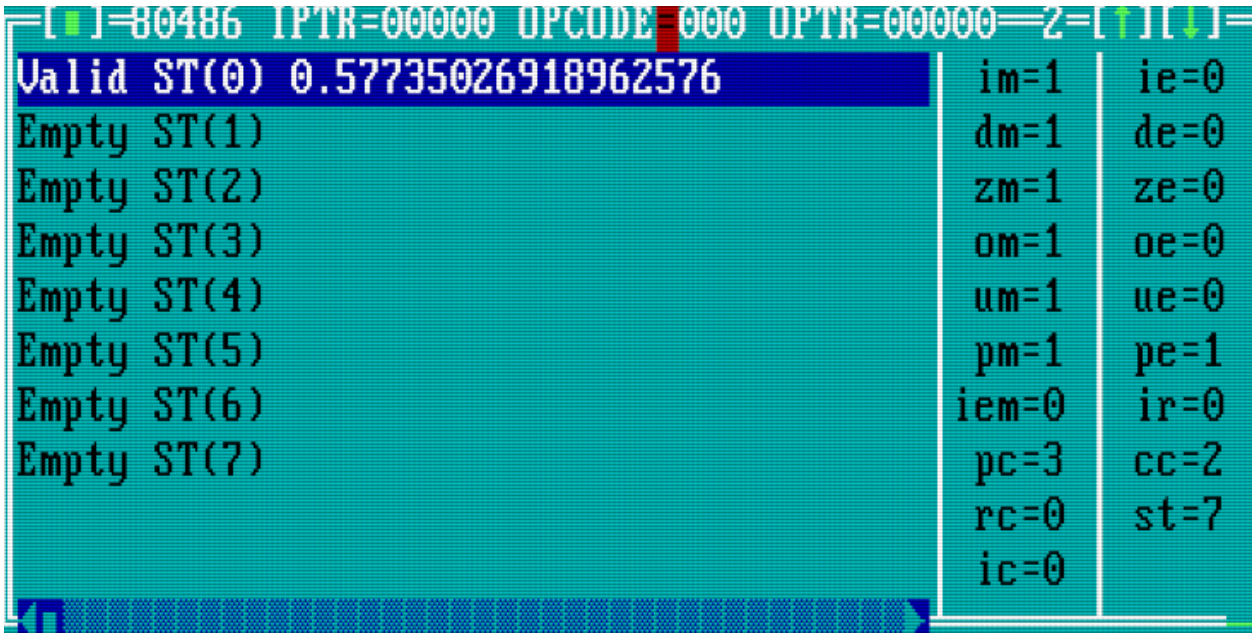
4. Compute sec(x), cosec(x) and cot(x).

Code:

=> Cot(x)

```
        data segment
        th dd 60.0
        d dd 180.0
        pi dd 3.1428
        res dd ?
        data ends
```

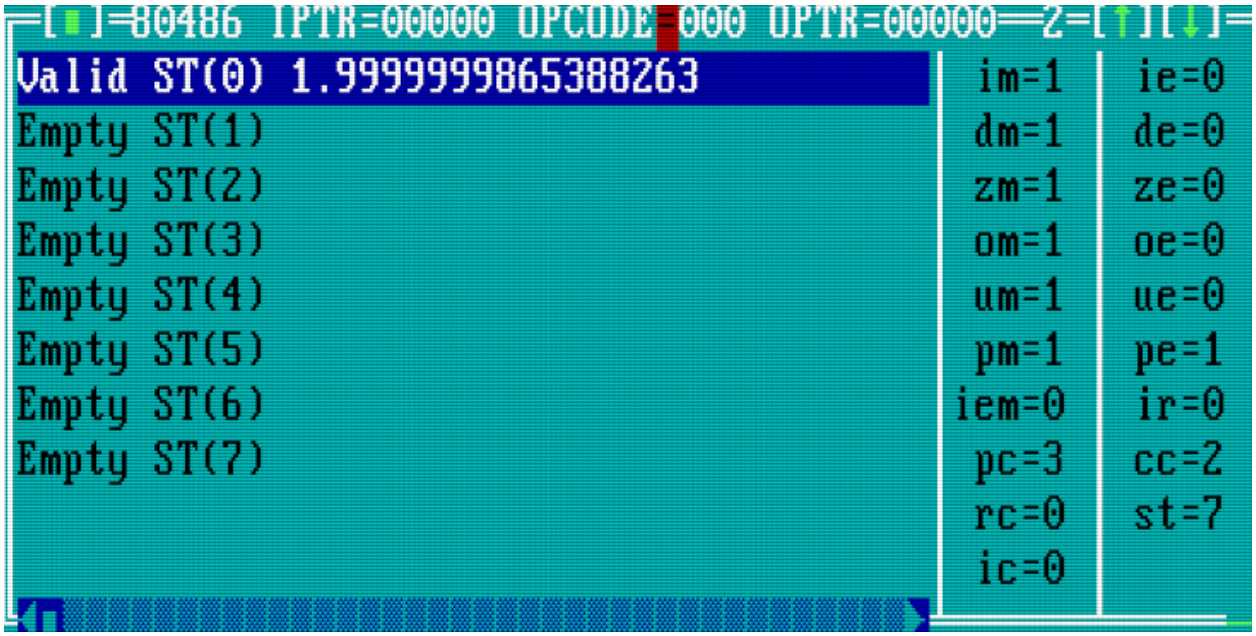
```
code segment


assume cs:code , ds:data
start:
mov ax, data
mov ds, ax
finit
fld th
fldpi
fmul
fdiv d
fptan
fxch
fdiv
fst res


int 03h
code ends
end start
```



=> Sec

```
data segment
th dd 60.0
```

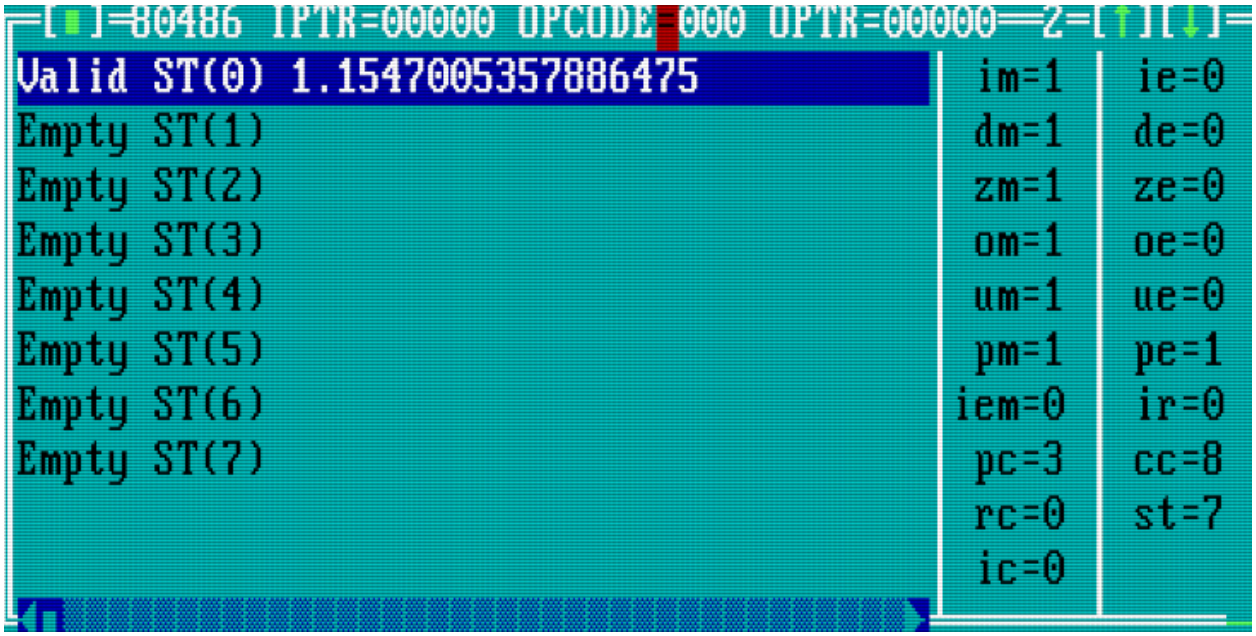
```
d dd 180.0


res dd ?
data ends


code segment


assume cs:code , ds:data
start:
mov ax, data
mov ds, ax
finit
fld th
fldpi
fmul
fdiv d
fptan
fxch
fst res


fmul res
fadd
fsqrt
int 03h
code ends
end start
```



=> **cosec**(x)

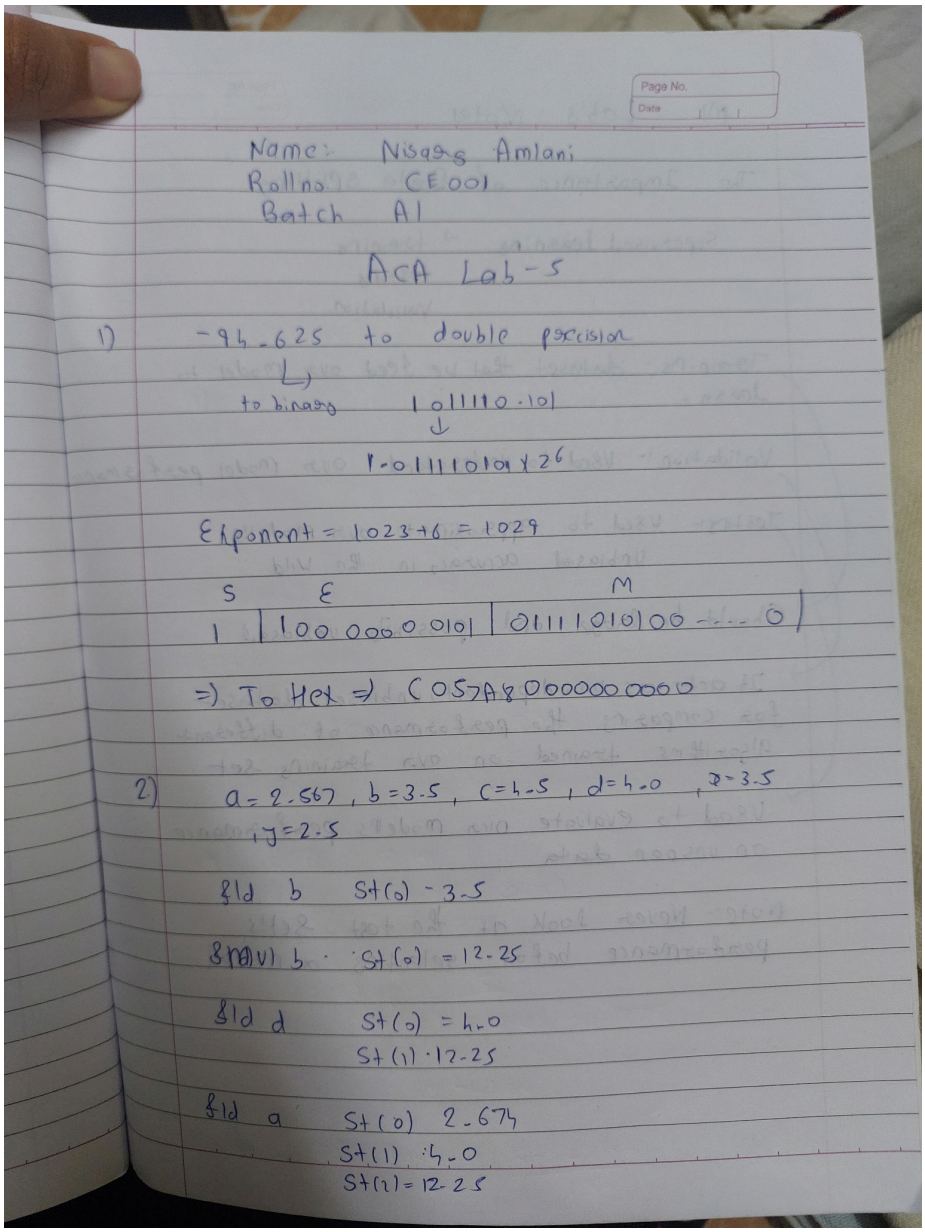
```
data segment
th dd 60.0
d dd 180.0
pi dd 3.142
one dd 1.0
res dd ?
data ends

code segment
assume cs:code , ds:data
start:
mov ax, data
mov ds, ax
finit
fld th
fldpi
fmul
fdiv d
fptan
fxch
fdiv
fst res
fmul res
```

```
fld one
fadd
fsqrt
fst res


int 03h
code ends
end start
```



Name : Nisarg .k.Amlani
Roll:Ce001
ID:22ceueg082

Lab :05



Name: Nisarg Amlani
Roll no. CE001
Batch A1

ACA Lab-5

1) -94.625 to double precision

↳ to binary 1011110.101

↓

1.011110101 x 2^6

Exponent = 1023+6 = 1029

S	E	M
1	10000000101	01111010100 0

=) To Hex =) C057A8000000000

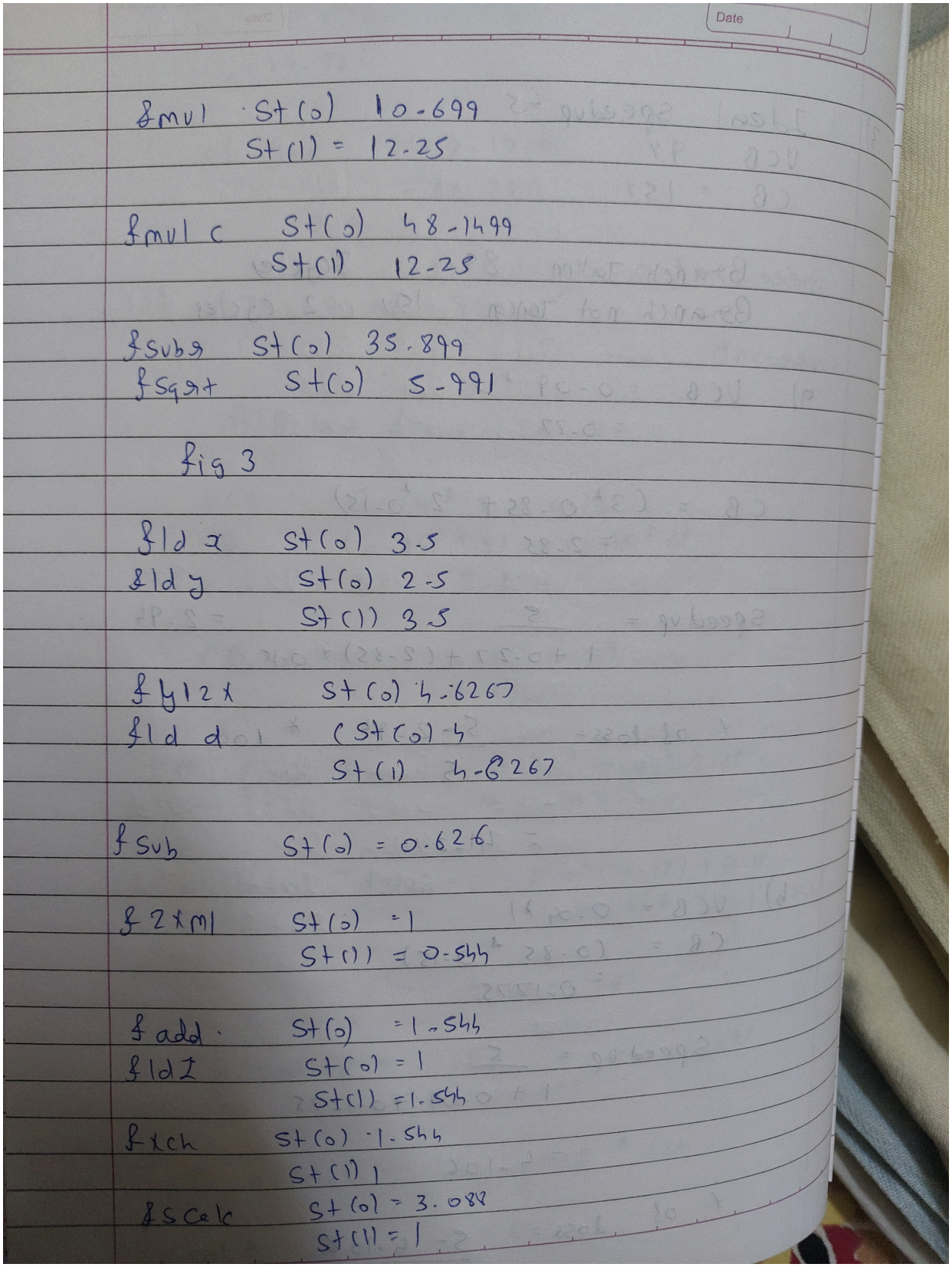
2) a = 2.567, b = 3.5, c = 4.5, d = 4.0, x = 3.5, y = 2.5

fld b St(0) - 3.5

fmul b St(0) = 12.25

fld d St(0) = 4.0
St(1) 12.25

fld a St(0) 2.674
St(1) 4.0
St(2) = 12.25



```
fmul    st(0)   10.699
        st(1) = 12.25

fmul c  st(0)   48.1499
        st(1)   12.25

fsubp   st(0)   35.899
fsqrt   st(0)   5.991
```

fig 3

```
fld x   st(0)   3.5
fld y   st(0)   2.5
        st(1)   3.5

fyl2x   st(0)   4.6267
fld d   (st(0)  4
        st(1)   4.6267

fsub    st(0) = 0.626

f2xm1   st(0)   =1
        st(1) = 0.544

fadd    st(0)   =1.544
fld1    st(0) = 1
        st(1) =1.544
fxch    st(0)  1.544
        st(1) 1
fscale  st(0) = 3.088
        st(1) = 1
```

3) Ideal Speedup = 5
UCB 9%
CB = 15%

Branch Taken 85% 3 cycles
Branch not Taken 15% 2 Cycles

a) UCB = 0.09 * 3
= 0.27

CB = (3 * 0.85 + 2 * 0.15)
= 2.85

Speed up = $\frac{5}{1 + 0.27 + (2.85) * 0.15}$ = 2.94

% of loss = $\frac{5 - 2.94}{5} * 100$

= 41.2%

b) UCB = 0.09 * 1
CB = (0.85 * 0.15)
= 0.1275

Speed up = $\frac{5}{1 + 0.09 + 0.1275}$

= 4.106

% of loss = $\frac{5 - 4.106}{} * 100$

= 17.88 %

% of gain = 41.2 - 17.88
= 22.29 %

c) BTB Found = 91 % → 0.96 Correct
→ 0.04 Incorrect

BTB not found = 9 %

UCB = 0.04 * 3 = 0.12
CB = (0.85 * 3) + (0.15 * 2)
= 2.85

CB delay = 2.85 * 0.09
= 0.2565

Misprediction = 0.91 * 0.04 * 2.85
= 0.10374

Total delay = 0.09(0.12) + 0.15
(0.2565 + 0.10374)

= 0.6479

Speed up = $\frac{5}{1 + 0.6479}$ = 4.695

% of loss = $\frac{5 - 4.695}{5} * 100$

= 6 %

Page No.
Date

4. Array

PE	PE	PE	PE	PE
LD1	LD2	LD3	LD4	LD5
Add1	Add2	Add3	Add4	Add5
Mul1	Mul2	Mul3	Mul4	Mul5
Sub1	Sub2	Sub3	Sub4	Sub5
DIV1	DIV2	DIV3	DIV4	DIV5
ST1	ST2	ST3	ST4	ST5

=) 6 cycles

Vector:

LD	ADD	MUL	SUB	DIV	ST
LD1					
LD2	ADD1				
LD3	ADD2	MUL1			
LD4	ADD3	MUL2	Sub1		
LD5	ADD4	MUL3	Sub2	DIV1	
	ADD5	MUL4	SUB3	DIV2	ST1
		MUL5	SUB4	DIV3	ST2
			SUB5	DIV4	ST3
				DIV5	ST4
					ST5

10 cycles

Ex1A : Hello World Program

```c
#include<stdio.h>

int main()
{
   int ID = 0;
   printf("hello(%d)",ID);
   printf("world(%d)",ID);

}
```

```
┌──(nisarg㉿fedora)-[~/Documents/pgms]
└─$ gcc -fopenmp 1.c

┌──(nisarg㉿fedora)-[~/Documents/pgms]
└─$ ./a.out
hello(0)world(0)
```

Ex 1B : Write a multithreaded program that prints “hello world”

```c
#include "omp.h"
#include <stdio.h>

int main()
{
   #pragma opm parallel
   {
       int ID = omp_get_thread_num();
       printf("hello(%d)",ID);
       printf("world(%d)",ID);
   }
}
```

```
hello(0)world(0)
hello(8)hello(1)hello(4)world(4)
hello(3)world(3)
hello(7)world(7)
world(8)
world(1)
hello(5)world(5)
hello(2)world(2)
hello(11)world(11)
hello(9)world(9)
hello(6)world(6)
hello(10)world(10)
```

Ex 1C : Write a multithreaded program to perform addition

```c
#include "omp.h"
#include <stdio.h>

int main()
{
   int sum = 0;
   int arr[100];
   for(int i = 0 ;i<100 ;i++)
   {
       arr[i] = 1;
   }

   #pragma omp parallel
   {
       int ID = omp_get_thread_num();
       for(int i = 0; i<100;i++)
       {
           sum += arr[i];
           printf("Sum = (%d)\n",sum);
       }
   }
```

```
}
```

Name : Nisarg .k. Amlani
Roll : Ce001
Id : 22ceueg082

Lab : 6

Code Pi using Serial

```
#include<stdio.h>
#include<stdlib.h>
#include "omp.h"

static long num_steps = 10000000;
double step;
int main(){
   int i ;
   double x  , pi , sum = 0.0;

   step = 1.0/(double)num_steps;
   double start = omp_get_wtime();

   for(i =0; i<num_steps; i++)
   {
       x = (i+0.5)*step;
       sum += 4.0/(1.0+x*x);
   }
   pi = step * sum;
   double end = omp_get_wtime();
   printf("Pi with OpenMP: %.15f\n",pi);
   printf("Time taken: %.15f seconds\n", end - start);
}
```

Output

```
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
Pi with OpenMP: 3.141592653589731
Time taken: 0.022761173000617 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
Pi with OpenMP: 3.141592653589731
Time taken: 0.023106741999982 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
Pi with OpenMP: 3.141592653589731
Time taken: 0.023597235999659 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
Pi with OpenMP: 3.141592653589731
Time taken: 0.023615467000127 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$
```

Code Pi using Parallel

```c
#include<stdlib.h>
#include<stdio.h>
#include "omp.h"

static long num_steps = 10000000;
double step;

#define NUM_THREADS 4

int main()
{
   int i ,nthrds;
   double pi , sum[NUM_THREADS];
   step = 1.0/(double) num_steps;
   omp_set_num_threads(NUM_THREADS);
   double start = omp_get_wtime();

   #pragma omp parallel
   {
```

```
        int i,id , nthreads;
        double x;
        id = omp_get_thread_num();
        nthreads = omp_get_num_threads();

        if(id == 0 ){ nthrds = nthreads;
        printf("threads: %d\n", nthreads);}
        for(i = id , sum[id] =0.0 ; i<num_steps;i = i+nthreads)
        {
            x = (i+0.5)*step;
            sum[id] += 4.0/(1.0+x*x);
        }
    }
    double end = omp_get_wtime();
    printf("nthreads: %d\n", nthrds);
    for(i = 0,pi = 0.0;i<nthrds;i++)
    {
        pi += step * sum[i];
    }
    printf("Pi with OpenMP: %.15f\n",pi);
    printf("Time taken: %.15f seconds\n", end - start);
}
```

Output

```
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Parallel Pi with OpenMP: 3.141592653589686
Time taken: 0.080148289000135 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Parallel Pi with OpenMP: 3.141592653589686
Time taken: 0.079596599000070 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Parallel Pi with OpenMP: 3.141592653589686
Time taken: 0.079434426999796 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Parallel Pi with OpenMP: 3.141592653589686
Time taken: 0.047810471000048 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$
```

Code Pi using Padding

```
#include<stdlib.h>
#include<stdio.h>
#include "omp.h"

static long num_steps = 10000000;
double step;

#define NUM_THREADS 4
#define PAD 8

int main()
{
   int i ,nthrds;
   double pi , sum[NUM_THREADS][PAD];
   step = 1.0/(double) num_steps;
   omp_set_num_threads(NUM_THREADS);
   double start = omp_get_wtime();

   #pragma omp parallel
   {
       int i,id , nthreads;
       double x;
       id = omp_get_thread_num();
       nthreads = omp_get_num_threads();

       if(id == 0 ){ nthrds = nthreads;
       printf("threads: %d\n", nthreads);}
       for(i = id , sum[id][0] =0.0 ; i<num_steps;i = i+nthreads)
       {
           x = (i+0.5)*step;
           sum[id][0] += 4.0/(1.0+x*x);
       }
   }
   double end = omp_get_wtime();

   for(i = 0,pi = 0.0;i<nthrds;i++)
   {
```

```
        pi += step * sum[i][0];
    }
    printf("Padding Pi with OpenMP: %.15f\n",pi);
    printf("Time taken: %.15f seconds\n", end - start);
}
```

Output

Code Pi using Critical

```
#include <stdlib.h>
#include <stdio.h>
#include "omp.h"

static long num_steps = 10000000;
double step;

#define NUM_THREADS 4

int main()
```

```
{
   int i, nthrds;
   double pi;
   step = 1.0 / (double)num_steps;
   omp_set_num_threads(NUM_THREADS);
   double start = omp_get_wtime();

#pragma omp parallel
   {
       int i, id, nthreads;
       double x, sum;
       id = omp_get_thread_num();
       nthreads = omp_get_num_threads();

       if (id == 0)
       {
           nthrds = nthreads;
           printf("threads: %d\n", nthreads);
       }
       for (i = id, sum = 0.0; i < num_steps; i = i + nthreads)
       {
           x = (i + 0.5) * step;
           sum += 4.0 / (1.0 + x * x);
       }

#pragma omp critical
       pi += step * sum;
   }
   double end = omp_get_wtime();

   printf("Critical Pi with OpenMP: %.15f\n", pi);
   printf("Time taken: %.15f seconds\n", end - start);
}
```

Output

```
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ gcc -fopenmp pi_critical.c
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
 threads: 4
 Critical Pi with OpenMP: 3.141592653589686
 Time taken: 0.007802002000062 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
 threads: 4
 Critical Pi with OpenMP: 3.141592653589686
 Time taken: 0.008252094999989 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
 threads: 4
 Critical Pi with OpenMP: 3.141592653589686
 Time taken: 0.007659092000722 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
 threads: 4
 Critical Pi with OpenMP: 3.141592653589686
 Time taken: 0.007902730999376 seconds
```

Code Pi using Atomic

```c
#include <stdlib.h>
#include <stdio.h>
#include "omp.h"

static long num_steps = 10000000;
double step;

#define NUM_THREADS 4

int main()
{
   int i, nthrds;
   double pi;
   step = 1.0 / (double)num_steps;
   omp_set_num_threads(NUM_THREADS);
   double start = omp_get_wtime();

#pragma omp parallel
```

```
    {
        int i, id, nthreads;
        double x, sum;
        id = omp_get_thread_num();
        nthreads = omp_get_num_threads();

        if (id == 0)
        {
            nthrds = nthreads;
            printf("threads: %d\n", nthreads);
        }
        for (i = id, sum = 0.0; i < num_steps; i = i + nthreads)
        {
            x = (i + 0.5) * step;
            sum += 4.0 / (1.0 + x * x);
        }
        // sum = sum *step;
#pragma atomic
        pi += step * sum;
    }
    double end = omp_get_wtime();

    printf("Atomic Pi with OpenMP: %.15f\n", pi);
    printf("Time taken: %.15f seconds\n", end - start);
}
```

Output

```
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Atomic Pi with OpenMP: 3.141592653589686
Time taken: 0.008723798000574 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Atomic Pi with OpenMP: 3.141592653589686
Time taken: 0.017979409999498 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Atomic Pi with OpenMP: 3.141592653589686
Time taken: 0.009777014999599 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Atomic Pi with OpenMP: 3.141592653589686
Time taken: 0.009044248000464 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$
```

Code Sum of Array using Serial

```
#include<stdio.h>
#include<stdlib.h>
#include "omp.h"

static long num_steps = 100;

int main(){
   int arr[num_steps]  , sum = 0;

   for(int i = 0; i < num_steps; i++)
   {
       arr[i] = i+1;
   }

   double start = omp_get_wtime();

   for(int i =0; i<num_steps; i++)
   {
       sum += arr[i];
   }

   double end = omp_get_wtime();
   printf("Serial Sum: %d\n", sum);
   printf("Time taken: %.15f seconds\n", end - start);
}
```

Output

```
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
Serial Sum: 5050
Time taken: 0.000000424999598 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
Serial Sum: 5050
Time taken: 0.000001315000191 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
Serial Sum: 5050
Time taken: 0.000000414999704 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$
```

Code Sum of Array using Parallel

```c
#include <stdio.h>
#include <stdlib.h>
#include "omp.h"

static long num_steps = 100;
#define NUM_THREADS 4

int main()
{
   int arr[num_steps], sum[NUM_THREADS] = {0}, nthrds;

   for (int i = 0; i < num_steps; i++)
   {
       arr[i] = i + 1;
   }

   omp_set_num_threads(NUM_THREADS);

   double start = omp_get_wtime();

#pragma omp parallel
   {
```

```
        int i = 0, id, nthreads;
        id = omp_get_thread_num();
        nthreads = omp_get_num_threads();
        if (id == 0)
        {
            nthrds = nthreads;
            printf("threads: %d\n", nthreads);
        }
        for (i = id; i < num_steps; i += nthreads)
        {
            sum[id] += arr[i];
        }
    }
    int total_sum = 0;
    for (int i = 0; i < nthrds; i++)
    {
        total_sum += sum[i];
    }

    double end = omp_get_wtime();
    printf("Parallel Sum: %d\n", total_sum);
    printf("Time taken: %.15f seconds\n", end - start);
}
```

Output

```
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Parallel Sum: 5050
Time taken: 0.000288320999971 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Parallel Sum: 5050
Time taken: 0.000278491999779 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Parallel Sum: 5050
Time taken: 0.000283990999378 seconds
```

Code Sum of Array using Padding

```
#include <stdio.h>
#include <stdlib.h>
#include "omp.h"

static long num_steps = 100;
#define NUM_THREADS 4
#define PAD 4

int main()
{
   int arr[num_steps], sum[NUM_THREADS][PAD] = {0}, nthrds;

   for (int i = 0; i < num_steps; i++)
   {
       arr[i] = i + 1;
   }

   omp_set_num_threads(NUM_THREADS);

   double start = omp_get_wtime();

#pragma omp parallel
   {
       int i = 0, id, nthreads;
       id = omp_get_thread_num();
       nthreads = omp_get_num_threads();
       if (id == 0)
       {
           nthrds = nthreads;
           printf("threads: %d\n", nthreads);
       }
       for (i = id; i < num_steps; i += nthreads)
       {
           sum[id][0] += arr[i];
       }
   }
   int total_sum = 0;
```

```
    for (int i = 0; i < nthrds; i++)
    {
        total_sum += sum[i][0];
    }

    double end = omp_get_wtime();
    printf("Padding Sum: %d\n", total_sum);
    printf("Time taken: %.15f seconds\n", end - start);
}
```

Output

```
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Padding Sum: 5050
Time taken: 0.000281087000076 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Padding Sum: 5050
Time taken: 0.000274871999864 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Padding Sum: 5050
Time taken: 0.000268377000793 seconds
```

Code Sum of Array using Critical

```
#include <stdio.h>
#include <stdlib.h>
#include "omp.h"

static long num_steps = 100;
#define NUM_THREADS 4


int main()
{
```

```
   int arr[num_steps], sum = 0, nthrds;

   for (int i = 0; i < num_steps; i++)
   {
       arr[i] = i + 1;
   }

   omp_set_num_threads(NUM_THREADS);

   double start = omp_get_wtime();

#pragma omp parallel
   {
       int i = 0, id, nthreads,partial_sum =0;
       id = omp_get_thread_num();
       nthreads = omp_get_num_threads();
       if (id == 0)
       {
           nthrds = nthreads;
           printf("threads: %d\n", nthreads);
       }
       for (i = id; i < num_steps; i += nthreads)
       {
           partial_sum += arr[i];
       }

       #pragma omp critical
       sum += partial_sum;
   }


   double end = omp_get_wtime();
   printf("Critical Sum: %d\n",sum);
   printf("Time taken: %.15f seconds\n", end - start);
}
```

Output

```
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Critical Sum: 5050
Time taken: 0.000273893000667 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Critical Sum: 5050
Time taken: 0.000227043000450 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Critical Sum: 5050
Time taken: 0.000317171000461 seconds
```

Code Sum of Array using Atomic

```c
#include <stdio.h>
#include <stdlib.h>
#include "omp.h"

static long num_steps = 100;
#define NUM_THREADS 4

int main()
{
   int arr[num_steps], sum = 0;

   for (int i = 0; i < num_steps; i++)
   {
       arr[i] = i + 1;
   }

   omp_set_num_threads(NUM_THREADS);

   double start = omp_get_wtime();

#pragma omp parallel
```

```
    {
        int id = omp_get_thread_num();
        int nthreads = omp_get_num_threads();

        if (id == 0)
        {
            printf("threads: %d\n", nthreads);
        }

        for (int i = id; i < num_steps; i += nthreads)
        {
#pragma omp atomic
            sum += arr[i]; // Atomic addition
        }
    }

    double end = omp_get_wtime();
    printf("Atomic Sum: %d\n", sum);
    printf("Time taken: %.15f seconds\n", end - start);

    return 0;
}
```

Output

```
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Atomic Sum: 5050
Time taken: 0.000282887000139 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Atomic Sum: 5050
Time taken: 0.000273079000181 seconds
user1@celab2-ThinkCentre-neo-50s-Gen-3:~/nisarg/lab6$ ./a.out
threads: 4
Atomic Sum: 5050
Time taken: 0.000293477999548 seconds
```

Name : Nisarg .k.Amlani
Roll : Ce001
Id : 22ceueg082

LAB : 07

---

1. Create a parallel version of the pi program using loop construct. Display the value of pi
and the run time and respective number of threads. (Run four instances of the program for
number of threads ranging from 1 to 4). Use following runtime library routines.
a. int omp_get_num_threads();
b. int omp_get_thread_num();
c. double omp_get_wtime();

```
#include <stdio.h>
#include "omp.h"

static long num_steps = 10000000;
double step;

#define NUM_THREADS 4

int main()
{
   int i, nthrds;
   double pi, sum[NUM_THREADS] = {0.0};
   step = 1.0 / (double)num_steps;
   omp_set_num_threads(NUM_THREADS);
   double start = omp_get_wtime();

#pragma omp parallel
```

```
    {
        int  id, nthreads;
        double x;
        id = omp_get_thread_num();
        nthreads = omp_get_num_threads();
        if (id == 0)
        {
            nthrds = nthreads;
        }

        #pragma omp for
        for (i = 0; i < num_steps; i++)
        {
            x = (i + 0.5) * step;
            sum[id] += 4.0 / (1.0 + x * x);
        }
    }


    double end = omp_get_wtime();
    printf("nthreads: %d\n", nthrds);
    for (i = 0, pi = 0.0; i < nthrds; i++)
    {
        pi += step * sum[i];
    }


    printf("Pi with OpenMP: %.15f\n", pi);
    printf("Time taken: %.15f seconds\n", end - start);
}
```

=> Outputs

```
┌──(nisarg㉿fedora)-[~/…/Sem-6/Sem_6_repo/ACA/Lab-7]
└─$ ./a.out
nthreads: 4
Pi with OpenMP: 3.141592653589670
Time taken: 0.314715906999481 seconds
```

```
┌──(nisarg㉿fedora)-[~/…/Sem-6/Sem_6_repo/ACA/Lab-7]
└─$ ./a.out
nthreads: 3
Pi with OpenMP: 3.141592653589728
Time taken: 0.261631736000709 seconds
```

```
┌──(nisarg㉿fedora)-[~/…/Sem-6/Sem_6_repo/ACA/Lab-7]
└─$ ./a.out
nthreads: 2
Pi with OpenMP: 3.141592653589923
Time taken: 0.204361284999322 seconds
```

```
┌──(nisarg㉿fedora)-[~/…/Sem-6/Sem_6_repo/ACA/Lab-7]
└─$ ./a.out
nthreads: 1
Pi with OpenMP: 3.141592653589731
Time taken: 0.060633639999651 seconds
```

2. Write an OpenMP program to perform matrix multiplication using loop construct.

```
#include <stdio.h>
#include <omp.h>

#define NUM_THREADS 4
#define ROW 3
#define COL 3

int main()
{
   int mat[ROW][COL] = {{1, 2, 3}, {1, 2, 3}, {1, 2, 3}};
   int mat2[ROW][COL] = {{1, 2, 3}, {1, 2, 3}, {1, 2, 3}};
   int result[ROW][COL] = {0};

   int nthrds = 0;
   omp_set_num_threads(NUM_THREADS);
   double start = omp_get_wtime();

#pragma omp parallel
   {
       int id = omp_get_thread_num();
       int nthreads = omp_get_num_threads();

#pragma omp single
       {
           nthrds = nthreads;
       }

       // Parallelized outer and middle loops
#pragma omp for collapse(2)
       for (int i = 0; i < ROW; i++)
       {
           for (int j = 0; j < COL; j++)
           {
```

```
                int sum = 0;
                for (int k = 0; k < COL; k++)
                {
                    sum += mat[i][k] * mat2[k][j];
                }
                result[i][j] = sum;  // Safe write, each thread
writes a separate result[i][j]
            }
        }
    }

    double end = omp_get_wtime();
    printf("Threads: %d\n", nthrds);
    printf("Time Taken: %.15f seconds\n", end - start);

    printf("\nResult Matrix:\n");
    for (int i = 0; i < ROW; i++)
    {
        for (int j = 0; j < COL; j++)
        {
            printf(" %d ", result[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

=> Output

```
┌──(nisarg㉿fedora)-[~/…/Sem-6/Sem_6_repo/ACA/Lab-7]
└─$ ./a.out
Threads: 4
Time Taken: 0.000181191999218 seconds

Result Matrix:
 6  12  18
 6  12  18
 6  12  18
```

3. Write an OpenMP program to compute the dot product of two vectors using the reduction clause to sum up partial results.

```c
#include <stdio.h>
#include "omp.h"

#define NUM_THREADS 4
#define num_steps 100

int main()
{
   long int prod = 0;
   int nthrds;
   int arr[num_steps], arr2[num_steps];

   for (int i = 0; i < num_steps; i++)
   {
       arr[i] = i + 1;
       arr2[i] = i + 1;
   }
```

```
   omp_set_num_threads(NUM_THREADS);
   double start = omp_get_wtime();

#pragma omp parallel
   {
       int partial_mul = 0;
       int id ;
       id = omp_get_thread_num();
       if(id == 0)
       {
           printf("Threads %d\n",omp_get_num_threads());
       }
#pragma omp for reduction(+ : prod)
       for (int i = 0; i < num_steps; i++)
       {
           partial_mul = arr[i] * arr2[i];
           prod += partial_mul;
       }
   }

   double end = omp_get_wtime();

   printf("Time Taken %.16f\n", end - start);
   printf("Dot Product %ld\n", prod);
}
```

=> Output

```
┌──(nisarg㉿fedora)-[~/…/Sem-6/Sem_6_repo/ACA/Lab-7]
└─$ ./a.out
Threads 4
Time Taken 0.0002298550007254
Dot Product 338350
```

4. Write an OpenMP program to demonstrate the difference between static and dynamic scheduling in OpenMP by summing elements of a large array.

```
#include <stdio.h>
#include "omp.h"

#define NUM_THREADS 4
#define ARR_SZ 1000000
#define CHUNK_SZ 1000

int main()
{
   int arr[ARR_SZ], sum_static = 0, sum_dynamic = 0;
   for (int i = 0; i < ARR_SZ; i++)
   {
       arr[i] = i + 1;
   }

   omp_set_num_threads(NUM_THREADS);
   double start_static = omp_get_wtime();

#pragma omp parallel
   {
       long int partial_sum = 0;

#pragma omp for schedule(static, CHUNK_SZ)
       for (int i = 0; i < ARR_SZ; i++)
       {
           partial_sum += arr[i];
       }

#pragma omp critical
       sum_static += partial_sum;
   }
```

```
    double end_static = omp_get_wtime();
    double start_dynamic = omp_get_wtime();

    omp_set_num_threads(NUM_THREADS);

#pragma omp parallel
    {
        long int partial_sum = 0;

#pragma omp for schedule(dynamic, CHUNK_SZ)
        for (int i = 0; i < ARR_SZ; i++)
        {
            partial_sum += arr[i];
        }

#pragma omp critical
        sum_dynamic += partial_sum;
    }

    double end_dynamic = omp_get_wtime();

    printf("Static Scheduling Sum: %ld, Time: %.6f\n", sum_static,
end_static - start_static);
    printf("Dynamic Scheduling Sum: %ld, Time: %.6f\n", sum_dynamic,
end_dynamic - start_dynamic);
}
```

=> Output

```
┌──(nisarg㉿fedora)-[~/…/Sem-6/Sem_6_repo/ACA/Lab-7]
└─$ ./a.out
Static Scheduling Sum: 1784293664, Time: 0.000876
Dynamic Scheduling Sum: 1784293664, Time: 0.000558
```

Name : Nisarg .k. Amlani
Roll : Ce001
Id : 22ceueg082

LAB :0 8

---

1. Write an OpenMP program using 4 threads, each thread calculates factorial of its id and
then all the factorials respective to threads need to be added to get on final sum. Use shared
and private clauses.
Output: Individual threads factorial result with their respective ids and the final sum of all
the factorials.

```c
#include<stdio.h>
#include<stdlib.h>
#include"omp.h"

int main(){
   int sum = 1, fact = 1;
   omp_set_num_threads(4);
   #pragma omp parallel private(fact) reduction(*:sum)
   {
       fact = 1;
       int id = omp_get_thread_num();
       int nthrds = omp_get_num_threads();
       for(int i = 1+id; i <= 6; i+=nthrds){
           fact *=  i;
       }
       printf("ID = %d, fact = %d\n",id,fact);
       sum *= fact;
   }
   printf("Sum = %d\n",sum);
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/lab-8/lab8]
└─$ ./a.out
ID = 2, fact = 3
ID = 0, fact = 5
ID = 3, fact = 4
ID = 1, fact = 12
Sum = 720
```

Name : Nisarg .k. Amlani
Roll : Ce001
Id : 22ceueg082

LAB: 09

**1. Write an OpenMP program using 4 threads, each thread calculates factorial of its id and
then all the factorials respective to threads need to be added to get on final sum. Use shared
and private clauses.
Output: Individual threads factorial result with their respective ids and the final sum of all
the factorials.**

```c
#include<stdio.h>
#include<stdlib.h>
#include"omp.h"

int main(){
   int sum = 1, fact = 1;
   omp_set_num_threads(4);
   #pragma omp parallel private(fact) reduction(*:sum)
   {
       fact = 1;
       int id = omp_get_thread_num();
       int nthrds = omp_get_num_threads();
       for(int i = 1+id; i <= 6; i+=nthrds){
           fact *=  i;
       }
       printf("ID = %d, fact = %d\n",id,fact);
       sum *= fact;
   }
   printf("Sum = %d\n",sum);
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/lab-8/lab8]
└─$ ./a.out
ID = 2, fact = 3
ID = 0, fact = 5
ID = 3, fact = 4
ID = 1, fact = 12
Sum = 720
```

2. Write an OpenMP program for the Pi program by making minimum changes in the serial
pi program that you can do.

```c
#include<stdio.h>
#include<stdlib.h>
#include "omp.h"

static long num_steps = 10000000;
double step;
int main(){
   int i ;
   double x   , pi , sum = 0.0;

   step = 1.0/(double)num_steps;
   double start = omp_get_wtime();

   #pragma omp parallel for private(x) reduction(+:sum)
   for(i =0; i<num_steps; i++)
   {
       x = (i+0.5)*step;
       sum += 4.0/(1.0+x*x);
   }
   pi = step * sum;
```

```
    double end = omp_get_wtime();
    printf("Pi with OpenMP: %.15f\n",pi);
    printf("Time taken: %.15f seconds\n", end - start);
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/lab-8/lab8]
└─$ ./a.out
Pi with OpenMP: 3.141592653589811
Time taken: 0.003733335001016 seconds
```

3. For the given C code for Mandelbrot set area computation.
a. Run the code multiple times and note the output.
b. Find the errors in the program.
c. Run the error free version.

```
/*
**  PROGRAM: Mandelbrot area
**
**  PURPOSE: Program to compute the area of a  Mandelbrot set.
**           Correct answer should be around 1.510659.
**           WARNING: this program may contain errors
**
**  USAGE:   Program runs without input ... just run the executable
**
**  HISTORY: Written:  (Mark Bull, August 2011).
**           Changed "comples" to "d_comples" to avoid collsion with
**           math.h complex type (Tim Mattson, September 2011)
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>
```

```
# define NPOINTS 1000
# define MAXITER 1000

void testpoint(void);

struct d_complex{
   double r;
   double i;
};

struct d_complex c;
int numoutside = 0;

int main(){
   int i, j;
   double area, error, eps  = 1.0e-5;
   omp_lock_t lock;
   omp_init_lock(&lock);
   omp_unset_lock(&lock);

   //   Loop over grid of points in the complex plane which contains the
Mandelbrot set,
   //   testing each point to see whether it is inside or outside the set.

   #pragma omp parallel for default(shared) private(c,eps)
   for (i=0; i<NPOINTS; i++) {
       for (j=0; j<NPOINTS; j++) {
           c.r = -2.0+2.5*(double)(i)/(double)(NPOINTS)+eps;
           c.i = 1.125*(double)(j)/(double)(NPOINTS)+eps;
           omp_set_lock(&lock);
               testpoint();
           omp_unset_lock(&lock);
       }
   }

   // Calculate area of set and error estimate and output the results


area=2.0*2.5*1.125*(double)(NPOINTS*NPOINTS-numoutside)/(double)(NPOINTS*N
POINTS);
```

```
   error=area/(double)NPOINTS;

   printf("Area of Mandlebrot set = %12.8f +/- %12.8f\n",area,error);
   printf("Correct answer should be around 1.510659\n");

}

void testpoint(void){

   // Does the iteration z=z*z+c, until |z| > 2 when point is known to be
outside set
   // If loop count reaches MAXITER, point is considered to be inside the
set

   struct d_complex z;
   int iter;
   double temp;

   z=c;
   for (iter=0; iter<MAXITER; iter++){
       temp = (z.r*z.r)-(z.i*z.i)+c.r;
       z.i = z.r*z.i*2+c.i;
       z.r = temp;
       if ((z.r*z.r+z.i*z.i)>4.0) {
       numoutside++;
       break;
       }
   }

}
```

```
┌──(nisarg㉿fedora)-[~/…/ACA/lab-8/lab8/aca_lab9]
└─$ ./a.out
Area of Mandlebrot set =   5.62500000 +/-   0.00562500
Correct answer should be around 1.510659
```

4. Create a program that executes two independent tasks in parallel using OpenMP's sections
Directive.

```c
#include<stdio.h>
#include<stdlib.h>
#include"omp.h"

void calc(int id){
   for(int i = 0 ; i < 10 ; i++)
       printf("Id = %d,  i = %d\n", id, i);
}

int main(){
   #pragma omp parallel
   {
       #pragma omp sections
       {
           #pragma omp section
               calc(omp_get_thread_num());
           #pragma omp section
               calc(omp_get_thread_num());
           #pragma omp section
               calc(omp_get_thread_num());
       }
   }
}
```

```
┌──(nisarg㉿fedora)-[~/…/ACA/lab-8/lab8/aca_lab9]
└─$ ./a.out
Id = 3,  i = 0
Id = 3,  i = 1
Id = 3,  i = 2
Id = 3,  i = 3
Id = 3,  i = 4
Id = 3,  i = 5
Id = 3,  i = 6
Id = 3,  i = 7
Id = 3,  i = 8
Id = 3,  i = 9
Id = 0,  i = 0
Id = 0,  i = 1
Id = 0,  i = 2
Id = 0,  i = 3
Id = 0,  i = 4
Id = 0,  i = 5
Id = 0,  i = 6
Id = 0,  i = 7
Id = 0,  i = 8
Id = 0,  i = 9
Id = 6,  i = 0
Id = 6,  i = 1
Id = 6,  i = 2
Id = 6,  i = 3
Id = 6,  i = 4
Id = 6,  i = 5
Id = 6,  i = 6
Id = 6,  i = 7
Id = 6,  i = 8
Id = 6,  i = 9

┌──(nisarg㉿fedora)-[~/…/ACA/lab-8/lab8/aca_lab9]
└─$
```

Name : Nisarg .k. Amlani
Roll : Ce001
Id : 22ceueg082

Lab : 10

---

1. Write header.h file and include process_fork() and process_join() and shared() functions.

```
#include<unistd.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<sys/shm.h>
#include<sys/ipc.h>
int process_fork (int nproc)
{
   int j;
   for (j=1;j<nproc; j++)
   {
       if (fork()==0)
       return (j);
   }
   return(0);
}

void process_join (int nproc, int id)
{
   int i;
   if (id==0)
   {
       for (i=1;i<nproc; i++)
           wait(0);
   }
   else
       exit(0);
}
```

```
char *shared (int size, int *shmid)
{
   *shmid =shmget(IPC_PRIVATE,size,0666|IPC_CREAT);
   return (shmat(*shmid,0,0));
}
```

2. WAP to create n number of processes.

```
#include <stdio.h>
#include "header.h"

int main()
{
   int id, nproc;

   printf("Enter number of processes you want to create: ");
   scanf("%d", &nproc);

   id = process_fork(nproc);

   printf("Process id is %d\n", id);

   process_join(nproc, id);

   printf("Parent id is %d\n", id);

   return 0;
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/ACA_10/ACA_10]
└─$ ./a.out
Enter number of processes you want to create: 5
Process id is 1
Process id is 2
Process id is 3
Process id is 0
Process id is 4
Parent id is 0

┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/ACA_10/ACA_10]
```

3. Write a parallel program to add four variables (a,b,c,d)

```c
#include "header.h"
#include<stdio.h>
#include<stdlib.h>
int main(){
  int id, nproc;
  int sum1 = 0, *sum2 , sum=0, shmid;
  sum2 = (int *)shared(sizeof(int),&shmid);
  *sum2 = 0;
  int a=1, b=10, c=9, d=20;
  id = process_fork(2);

  if(id==0){
      sum1 = a+b;
      printf("sum by Parent process : %d\n", sum1);
  }
  else{
      *sum2 = c+d;
      printf("sum by process %d  : %d\n", id, *sum2);
  }
  process_join(2,id);
  sum = sum1 + *sum2;
```

```
  printf("Final sum %d\n", sum);
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/ACA_10/ACA_10]
└─$ ./a.out
sum by Parent process : 11
sum by process 1  : 29
Final sum 40

┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/ACA_10/ACA_10]
```

4. Write a parallel program to copy one array into another using loop splitting.

```
#include "header.h"
#include <stdio.h>
int main()
{
 int nproc = 4;
 int id, a[10], *b, i;
 b = (int *)shared(sizeof(i) * 10, &i);
 for (i = 0; i < 10; i++)
 {
     printf ("Enter elements of a [%d]:",i+1);
     scanf ("%d", &a [i]);
 }
 id = process_fork(nproc);
 if (id == 0)
 {
     for (i = 0; i < 10; i += nproc){
         printf("Element %d is copied by process %d\n", i, id);
         *(b + i) = a[i];
     }
```

```
}
else if (id == 1)
{
    for (i = 1; i <= 10; i += nproc){
        printf("Element %d is copied by process %d\n", i, id);
        *(b + i) = a[i];
    }
}
else if (id == 2)
{
    for (i = 2; i < 10; i += nproc){
        printf("Element %d is copied by process %d\n", i, id);
        *(b + i) = a[i];
    }
}
else
{
    for (i = 3; i <= 10; i += nproc){
        printf("Element %d is copied by process %d\n", i, id);
        *(b + i) = a[i];
    }
}


process_join(nproc, id);
printf("elements of b :");
for(int i=0;i<10;i++){
    printf("%d, ", b[i]);
}
printf("\n");
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/ACA_10/ACA_10]
└─$ ./a.out
Enter elements of a [1]:1
Enter elements of a [2]:2
Enter elements of a [3]:3
Enter elements of a [4]:4
Enter elements of a [5]:5
Enter elements of a [6]:6
Enter elements of a [7]:7
Enter elements of a [8]:8
Enter elements of a [9]:9
Enter elements of a [10]:1
Element 1 is copied by process 1
Element 5 is copied by process 1
Element 9 is copied by process 1
Element 2 is copied by process 2
Element 6 is copied by process 2
Element 0 is copied by process 0
Element 4 is copied by process 0
Element 8 is copied by process 0
Element 3 is copied by process 3
Element 7 is copied by process 3
elements of b :1, 2, 3, 4, 5, 6, 7, 8, 9, 1,

┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/ACA_10/ACA_10]
```

5. Write a parallel program to copy one array into another using loop splitting
and strictly using one loop.

```c
#include "header.h"
#include <stdio.h>
int main()
{
   int nproc = 4;
   int id, a[10], *b, i;
   b = (int *)shared(sizeof(i) * 10, &i);
   for (i = 0; i < 10; i++)
   {
       printf("Enter elements of a [%d]:", i + 1);
       scanf("%d", &a[i]);
```

```
    }
    id = process_fork(nproc);
    for (int i = id; i < 10; i += nproc)
    {
        printf("Element %d is copied by process %d\n", i, id);
        *(b + i) = a[i];
    }

    process_join(nproc, id);
    printf("elements of b :");
    for (int i = 0; i < 10; i++)
    {
        printf("%d, ", b[i]);
    }
    printf("\n");
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/ACA_10/ACA_10]
└─$ ./a.out
Enter elements of a [1]:1
Enter elements of a [2]:2
Enter elements of a [3]:3
Enter elements of a [4]:4
Enter elements of a [5]:5
Enter elements of a [6]:6
Enter elements of a [7]:7
Enter elements of a [8]:8
Enter elements of a [9]:9
Enter elements of a [10]:1
Element 1 is copied by process 1
Element 5 is copied by process 1
Element 9 is copied by process 1
Element 2 is copied by process 2
Element 6 is copied by process 2
Element 0 is copied by process 0
Element 4 is copied by process 0
Element 8 is copied by process 0
Element 3 is copied by process 3
Element 7 is copied by process 3
elements of b :1, 2, 3, 4, 5, 6, 7, 8, 9, 1,

┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/ACA_10/ACA_10]
```

6. Write a parallel program to do matrix addition with loop splitting.

```c
#include "header.h"
#include <stdio.h>
int main()
{
  // Declaration and scan values
  int id, a[3][3], b[3][3], *c, i, j, nproc = 3, shmid;
  c = (int *)shared(sizeof(int) * 3 * 3, &shmid);
  // c is shared among parent and children
  for (j = 0; j < 3; j++) // Scan matrix A
  {
      for (i = 0; i < 3; i++)
```

```
      {
          printf ("Enter element a [%d] [%d]:",j+1,i+1);
          scanf ("%d", &a [j] [i]);
      }
  }
  printf("\n");

  for (j = 0; j < 3; j++) // Scan matrix B
  {
      for (i = 0; i < 3; i++)
      {
          printf("Enter element b [%d] [%d]= ", j + 1, i + 1);
          scanf("%d", &b[j][i]);
      }
  }

  // program logic
  id = process_fork(nproc); // nproc=3
  for (i = id; i < 3; i += 3)
  {
      for (j = 0; j < 3; j++){
          *(c + 3 * i + j) = a[i][j] + b[i][j];
          printf("Element %d is added using process id: %d\n", *(c+3*i+j),
id);
      }
  }

  process_join(nproc, id); // Children are exited
   // Print final Values
  for (i = 0; i < 3; i++) // print matrix A
  {
      for (j = 0; j < 3; j++)
      {
          printf("a [%d][%d]=%d \t",i+1,j+1, a[i][j]);
      }
      printf("\n");
  }
  printf("\n");

  for (i = 0; i < 3; i++) // print matrix B
```

```
  {
      for (j = 0; j < 3; j++)
      {
          printf("b [%d][%d]=%d \t",i+1,j+1, b[i][j]);
      }
      printf("\n");
  }
  printf("\n");
   for (i = 0; i < 3; i++) // print resultant matrix C
  {
      for (j = 0; j < 3; j++)
          printf("c [%d][%d]=%d\t", i + 1, j+1, *(c + 3 * i + j));
      printf("\n");
  }
}
```

```
a [1][1]=1      a [1][2]=2      a [1][3]=3
a [2][1]=4      a [2][2]=5      a [2][3]=6
a [3][1]=7      a [3][2]=8      a [3][3]=9

b [1][1]=1      b [1][2]=2      b [1][3]=3
b [2][1]=4      b [2][2]=5      b [2][3]=6
b [3][1]=7      b [3][2]=8      b [3][3]=9

c [1][1]=2      c [1][2]=4      c [1][3]=6
c [2][1]=8      c [2][2]=10     c [2][3]=12
c [3][1]=14     c [3][2]=16     c [3][3]=18

┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/ACA_10/ACA_10]
└─$
```

7. Write a parallel program to do matrix multiplication with loop splitting.

```
#include "header.h"
#include <stdio.h>
```

```
int main()
{
   int m = 3;
   int mat1[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
   int mat2[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
   int id, nproc = 3, shmid, i;
   int *mat3;

   // Allocate shared memory for the result matrix (mat3)
   mat3 = (int *)shared(sizeof(int) * m * m, &shmid);

   // Print mat1
   printf("Matrix 1:\n");
   for(int i = 0; i < m; i++) {
       for(int j = 0; j < m; j++) {
           printf("%d ", mat1[i][j]);
       }
       printf("\n");
   }

   // Print mat2
   printf("Matrix 2:\n");
   for(int i = 0; i < m; i++) {
       for(int j = 0; j < m; j++) {
           printf("%d ", mat2[i][j]);
       }
       printf("\n");
   }

   // Fork processes for matrix multiplication
   id = process_fork(nproc);

   // Perform matrix multiplication in parallel
   for(i = id; i < m; i += nproc) {
       for(int j = 0; j < m; j++) {
           int sum = 0;
           for(int k = 0; k < m; k++) {
               sum += mat1[i][k] * mat2[k][j];
           }
           *(mat3 + (m * i) + j) = sum;
```

```
            printf("Process %d : %d\n", id, *(mat3 + (m * i) + j));
        }
    }

    // Join processes after multiplication
    process_join(nproc, id);

    // Print final result matrix mat3
    printf("\nFinal Matrix 3 (Result):\n");
    for(int i = 0; i < m; i++) {
        for(int j = 0; j < m; j++) {
            printf("%d ", *(mat3 + (m * i) + j));
        }
        printf("\n");
    }

    return 0;
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/ACA_10/ACA_10]
└─$ ./a.out
Matrix 1:
1 2 3
4 5 6
7 8 9
Matrix 2:
1 2 3
4 5 6
7 8 9
Process 1 : 66
Process 1 : 81
Process 1 : 96
Process 0 : 30
Process 0 : 36
Process 0 : 42
Process 2 : 102
Process 2 : 126
Process 2 : 150

Final Matrix 3 (Result):
30 36 42
66 81 96
102 126 150

┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/ACA_10/ACA_10]
└─$
```

8. Write a parallel program to find factorial of a number using loop splitting.

```c
#include <stdio.h>
#include "header.h"
int main(){
 int num;
 printf("Enter a number: ");
 scanf("%d", &num);
 int id, nproc = 4, shmid, i;
 int *fac, final_factorial = 1;
 fac = (int *)shared(sizeof(int)*nproc, &shmid);
 id = process_fork(nproc);
 for(i = id; i<nproc; i+=nproc)
```

```
{
    *(fac + i) = 1;
}
for(i = id +1; i<=num; i += nproc)
{
    *(fac + id) *= i;
}
printf("Process %d : %d\n", id, *(fac + id));
process_join(nproc, id);
for(i = 0; i<nproc; i++)
{
    final_factorial *= *(fac + i);
}
 printf("Factorial of %d is %d\n", num, final_factorial);
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/ACA_10/ACA_10]
└─$ ./a.out
Enter a number: 5
Process 1 : 2
Process 2 : 3
Process 0 : 5
Process 3 : 4
Factorial of 5 is 120

┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/ACA_10/ACA_10]
└─$
```

Name : Nisarg .k. Amlani
Roll : Ce001
Id : 22ceueg082

Lab : 11

---

1. To the header.h file created in LAB 10 include spin_lock_init() and spin_lock() and spin_unlock() functions.

```c
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/sem.h>

int process_fork(int nproc)
{

   int j;
   for (j = 1; j < nproc; j++)
   {
       if (fork() == 0)
           return (j);
   }

   return (0);
}

void process_join(int nproc, int id)
{

   int i;
```

```
    if (id == 0)
    {
        for (i = 1; i < nproc; i++)
            wait(0);
    }
    else
        exit(0);
}

char *shared(int size, int *shmid)
{

    *shmid = shmget(IPC_PRIVATE, size, 0666 | IPC_CREAT);
    return (shmat(*shmid, 0, 0));
}

void spin_lock_init(int *lock, int *condition)
{

    int control;
    *lock = semget(IPC_PRIVATE, 1, 0666 | IPC_CREAT);
    if (*condition == 1)
        control = 0;
    else
        control = 1;

    semctl(*lock, 0, SETVAL, control);
}

void spin_lock(int *lock)
{

    struct sembuf operations;
    operations.sem_num = 0;
    operations.sem_op = -1;
    operations.sem_flg = 0;
    semop(*lock, &operations, 1);
}

void spin_unlock(int *lock)
```

```
{

   struct sembuf operations;
   operations.sem_num = 0;
   operations.sem_op = 1;
   operations.sem_flg = 0;
   semop(*lock, &operations, 1);
}
```

2. WAP to add constant to array using self-scheduling.

```
#include "header.h"

int main(){
   int shmid;
   int *arr = (int*)shared(10*sizeof(int),&shmid);
   int n = 10;
   for(int i = 0 ; i < n ; i++)
       arr[i] = i;
   int n_proc = 4;
   int i = 0;
   int *index = (int*)shared(sizeof(int),&shmid);
   *index = 0;

   int id = process_fork(n_proc);
   while(1){
       i = *index;
       (*index)++;
       if(i >= n)
           break;
       arr[i] = arr[i] + 1;
          }
   process_join(n_proc,id);

   for(i = 0 ; i < n ; i++)
       printf("%d ",arr[i]);
   printf("\n");
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/aca_lab11/aca_lab11]
└─$ ./a.out
1 2 3 4 5 6 7 8 9 10

┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/aca_lab11/aca_lab11]
└─$
```

3. Write a parallel program to implement program 2 with locking

```c
#include <stdio.h>
#include "header.h"
int main()
{
   int *a, *next_index, i, id, k = 4, nproc = 3, shmid;
   int *lock1, unlock = 0;
   next_index = (int *)shared(sizeof(int), &shmid);
   *next_index = 0;
   a = (int *)shared(sizeof(int) * 10, &shmid);
   lock1 = (int *)shared(sizeof(int) * 10, &shmid);
   spin_lock_init(lock1, &unlock);
   for (int i = 0; i < 10; i++)
   {

       *(a + i) = i;
   }
   id = process_fork(nproc);
   while (1)
   {
       spin_lock(lock1);
       printf("process %d entered\n", i);
       i = *next_index;
       *next_index = *next_index + 1;
       printf("process %d exiting\n", i);
       spin_unlock(lock1);
       if (i < 10)
       {
           *(a + i) = *(a + i) + k;
```

```
        }
        else
            break;
    }
    process_join(nproc, id);
    for (int i = 0; i < 10; i++)
    {
        printf("%d \t", *(a + i));
    }
    printf("\n");
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/aca_lab11/aca_lab11]
└─$ ./a.out
process 32766 entered
process 0 exiting
process 0 entered
process 1 exiting
process 1 entered
process 2 exiting
process 32766 entered
process 3 exiting
process 2 entered
process 4 exiting
process 3 entered
process 5 exiting
process 4 entered
process 6 exiting
process 5 entered
process 7 exiting
process 32766 entered
process 8 exiting
process 6 entered
process 9 exiting
process 7 entered
process 10 exiting
process 8 entered
process 11 exiting
process 9 entered
process 12 exiting
4       5       6       7       8       9       10      11      12

┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/aca_lab11/aca_lab11]
└─$
```

4. Write a parallel program to calculate sum of 1 to n numbers using m processes.

```
#include "header.h" //Declaration and scan values
int main()
{
   int sum = 0, *final_sum, id, N, i, shmid, nproc = 4;
   int locked = 1, unlocked = 0, *lock;
   lock = (int *)shared(sizeof(int), &shmid);
   final_sum = (int *)shared(sizeof(sum), &i);
   *final_sum = 0;
   printf("\n Enter N :");
   scanf("%d", &N);
   spin_lock_init(lock, &unlocked);
   id = process_fork(nproc);
   printf("process %d does ->  ", id);
   for (i = id; i <= N; i += nproc)
   {
       sum = sum + i;
       printf("%d  ", i);
   }
   printf("\n");
   spin_lock(lock);
   *final_sum = sum + *final_sum;
   printf("id = %d sum = %d \n", id, sum);
   spin_unlock(lock);
   process_join(4, id);
   printf("\nSum: %d\n", *final_sum);
   return 0;
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/aca_lab11/aca_lab11]
└─$ ./a.out

 Enter N :8
process 1 does ->  1  5
id = 1 sum = 6
process 2 does ->  2  6
id = 2 sum = 8
process 0 does ->  0  4  8
id = 0 sum = 12
process 3 does ->  3  7
id = 3 sum = 10

Sum: 36

┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/aca_lab11/aca_lab11]
└─$
```

5. Write a parallel program to calculate sum of array of n elements using self-
scheduling.

```c
#include "header.h"
int main()
{
   int sum = 0, *final_sum, id, a[10], i = 0, nproc = 4;
   int unlocked = 0, *lock1, *lock2;
   int shmid, *index;
   final_sum = (int *)shared(sizeof(int), &shmid);
   *final_sum = 0;
   lock1 = (int *)shared(sizeof(int), &shmid);
   lock2 = (int *)shared(sizeof(int), &shmid);
   index = (int *)shared(sizeof(int), &shmid);
   *index = 0;
   for (i = 0; i < 10; i++)
   {
       a[i] = i;
   }
   spin_lock_init(lock1, &unlocked);
   spin_lock_init(lock2, &unlocked);
   id = process_fork(nproc);
   printf("process %d sums ", id);
```

```
    while (1)
    {
        spin_lock(lock1);
        i = *index;
        *index = *index + 1;
        printf("a[%d], ", i);
        spin_unlock(lock1);
        printf("\n");
        if (i < 10)
            sum = sum + a[i];
        else
            break;
    }
    spin_lock(lock2);
    *final_sum = sum + *final_sum;
    spin_unlock(lock2);
    process_join(nproc, id);
    printf("\n Sum: %d\n", *final_sum);
    return 0;
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/aca_lab11/aca_lab11]
└─$ ./a.out
process 1 sums a[0],
a[1],
a[2],
a[3],
a[4],
a[5],
a[6],
a[7],
a[8],
a[9],
a[10],
process 2 sums a[11],
process 0 sums a[12],
process 3 sums a[13],
```

6. Implementation of histogram in different ways

i. create histogram using self-scheduling

```
#include "header.h"
#include <stdio.h>
#define arrSize 15
int main()
{
   int a[arrSize];
   int i, *index, NoOfBins = 5, binSize, *histogram, *lock1, *lock2,
unlocked = 0, locked = 1;
   int shmidindex, shmidlock1, shmidlock2, shmidhist;
   int id, nproc, bin;
   int amin, amax;
   for (int i = 0; i < arrSize; i++)
   {
       // printf("enter a[%d]",i);
       a[i] = i;
   }
   amin = amax = a[0];
   for (i = 1; i < arrSize; i++)
   {
       if (amin > a[i])
           amin = a[i];
       if (amax < a[i])
           amax = a[i];
   }
   binSize = (amax - amin) / NoOfBins;
   index = (int *)shared(sizeof(int), &shmidindex);
   lock1 = (int *)shared(sizeof(int), &shmidlock1);
   lock2 = (int *)shared(sizeof(int), &shmidlock2);
   histogram = (int *)shared(sizeof(int) * NoOfBins, &shmidhist);
   printf("Bin Size : %d\n", binSize);
   printf("No. Of Bins : %d\n", NoOfBins);
   spin_lock_init(lock1, &unlocked);
   spin_lock_init(lock2, &unlocked);
   *index = 0;
   for (i = 0; i < NoOfBins; i++)
```

```
        *(histogram + i) = 0;
    nproc = NoOfBins;
    id = process_fork(nproc);
    while (1)
    {
        spin_lock(lock1);
        i = *index;
        *index = *index + 1;
        spin_unlock(lock1);
        if (i >= arrSize)
            break;
        bin = abs((a[i] - amin) / binSize);
        if (bin >= NoOfBins)
            bin = NoOfBins - 1;
        printf("Number %d is : %d\t Bin : %d\n", i, a[i], bin);
        spin_lock(lock2);
        *(histogram + bin) += 1;
        spin_unlock(lock2);
    }
    process_join(nproc, id);
    for (i = 0; i < NoOfBins; i++)
        printf("No of Items in Bin (%d): %d \n", i, *(histogram + i));
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/aca_lab11/aca_lab11]
└─$ ./a.out
Bin Size : 2
No. Of Bins : 5
Number 0 is : 0  Bin : 0
Number 1 is : 1  Bin : 0
Number 2 is : 2  Bin : 1
Number 3 is : 3  Bin : 1
Number 4 is : 4  Bin : 2
Number 5 is : 5  Bin : 2
Number 6 is : 6  Bin : 3
Number 7 is : 7  Bin : 3
Number 8 is : 8  Bin : 4
Number 9 is : 9  Bin : 4
Number 10 is : 10        Bin : 4
Number 11 is : 11        Bin : 4
Number 13 is : 13        Bin : 4
Number 14 is : 14        Bin : 4
Number 12 is : 12        Bin : 4
No of Items in Bin (0): 2
No of Items in Bin (1): 2
No of Items in Bin (2): 2
No of Items in Bin (3): 2
No of Items in Bin (4): 7
```

ii. implement histogram using loop splitting

```c
#include <stdio.h>
#include <stdlib.h>
#include "header.h"
#define arrSize 15
#define NoOfBins 5
void main()
{
   int a[arrSize];
   int binsize;
   int *histogram;
   int *lock, unlocked = 0, locked = 1;
   int shmidlock, shmidhist;
   int id, nproc;
   int bin;
   int amin, amax;
```

```
int i;
for (i = 0; i < arrSize; i++)
{
    a[i] = i + 23;
}
amin = a[0];
amax = a[0];
for (i = 1; i < arrSize; i++)
{
    if (amin > a[i])
        amin = a[i];
    if (amax < a[i])
        amax = a[i];
}
binsize = (amax - amin) / NoOfBins;
lock = (int *)shared(sizeof(int) * NoOfBins, &shmidlock);
histogram = (int *)shared(sizeof(int) * NoOfBins, &shmidhist);
printf("Bin Size: %d\n", binsize);
printf("No. Of Bins: %d\n", NoOfBins);
for (i = 0; i < NoOfBins; i++)
{
    spin_lock_init(lock + i, &unlocked);
    *(histogram + i) = 0;
}
printf("\n");
nproc = NoOfBins;
id = process_fork(nproc);
for (i = id; i < arrSize; i = i + nproc)
{
    bin = abs(a[i] - amin) / binsize;
    if (bin >= NoOfBins)
        bin = NoOfBins - 1;
    printf("Process(%d): Number [%d] is %d \t Bin: %d\n", id, i, a[i],
           bin);
    spin_lock(lock + bin);
    *(histogram + bin) += 1;
    spin_unlock(lock + bin);
}
printf("\n");
process_join(NoOfBins, id);
```

```
    for (i = 0; i < NoOfBins; i++)
    {
        printf("No Of Items in Bin (%d) is %d\n", i, *(histogram + i));
    }
}
```

```
┌──(nisarg㉿fedora)-[~/…/Sem_6_repo/ACA/aca_lab11/aca_lab11]
└─$ ./a.out
Bin Size: 2
No. Of Bins: 5

Process(1): Number [1] is 24     Bin: 0
Process(1): Number [6] is 29     Bin: 3
Process(1): Number [11] is 34    Bin: 4

Process(2): Number [2] is 25     Bin: 1
Process(2): Number [7] is 30     Bin: 3
Process(2): Number [12] is 35    Bin: 4

Process(3): Number [3] is 26     Bin: 1
Process(0): Number [0] is 23     Bin: 0
Process(0): Number [5] is 28     Bin: 2
Process(0): Number [10] is 33    Bin: 4
Process(3): Number [8] is 31     Bin: 4

Process(3): Number [13] is 36    Bin: 4

Process(4): Number [4] is 27     Bin: 2
Process(4): Number [9] is 32     Bin: 4
Process(4): Number [14] is 37    Bin: 4

No Of Items in Bin (0) is 2
No Of Items in Bin (1) is 2
No Of Items in Bin (2) is 2
No Of Items in Bin (3) is 2
No Of Items in Bin (4) is 7
```

iii. implement using partial histogram

```
#include <stdio.h>
#include <stdlib.h>
#include "header.h"
#define arrSize 15
int main()
{
   int a[arrSize];
   int NoOfBins;
   int binsize;
   int *histogram;
   int parhist[NoOfBins];
   int *lock, unlocked = 0, locked = 1;
   int shmidlock, shmidhist, shmidparhist;
   int id, nproc;
   int bin;
   int amin, amax;
   int i;
   printf("Enter No OF Bins: ");
   scanf("%d", &NoOfBins);
   // Initialize an Array
   for (i = 0; i < arrSize; i++)
   {
       // printf("Enter a[%d]: ", i);
       // scanf("%d", &a[i]);
       a[i] = i;
   }
   amin = a[0];
   amax = a[0];
   for (i = 1; i < arrSize; i++)
   {
       if (amin > a[i])
           amin = a[i];
       if (amax < a[i])
           amax = a[i];
   }
   binsize = (amax - amin) / NoOfBins;
   lock = (int *)shared(sizeof(int) * NoOfBins, &shmidlock);
   histogram = (int *)shared(sizeof(int) * NoOfBins, &shmidhist);
```

```
    printf("\nBin Size: %d", binsize);
    printf("\nNo. Of Bins: %d\n", NoOfBins);
    // Initialize spin_locks
    for (i = 0; i < NoOfBins; i++)
        spin_lock_init(lock + i, &unlocked);
    // Initialize histogram
    for (i = 0; i < NoOfBins; i++)
        *(histogram + i) = 0;
    printf("\n");
    nproc = NoOfBins;
    id = process_fork(nproc);
    for (i = 0; i < NoOfBins; i++)
        parhist[i] = 0;
    for (i = id; i < arrSize; i += nproc)
    {
        bin = abs(a[i] - amin) / binsize;
        if (bin >= NoOfBins)
            bin = NoOfBins - 1;
        printf("Process(%d): Number [%d] is %d\tBin: %d\n", id, i, a[i],
               bin);
        parhist[bin] += 1;
    }
    for (i = 0; i < NoOfBins; i++)
    {
        spin_lock(lock + i);
        *(histogram + i) += parhist[i];
        spin_unlock(lock + i);
    }
    printf("\n");
    process_join(nproc, id);
    for (i = 0; i < NoOfBins; i++)
    {
        printf("No Of Items in Bin (%d) is %d\n", i, *(histogram + i));
    }
    return 0;
}
```

```
Enter No Of Bins: 10

Bin Size: 1
No. Of Bins: 10

Process(1): Number [1] is 1     Bin: 1
Process(1): Number [11] is 11   Bin: 9

Process(2): Number [2] is 2     Bin: 2
Process(2): Number [12] is 12   Bin: 9

Process(3): Number [3] is 3     Bin: 3
Process(3): Number [13] is 13   Bin: 9

Process(4): Number [4] is 4     Bin: 4
Process(4): Number [14] is 14   Bin: 9

Process(5): Number [5] is 5     Bin: 5

Process(6): Number [6] is 6     Bin: 6

Process(0): Number [0] is 0     Bin: 0
Process(0): Number [10] is 10   Bin: 9

Process(7): Number [7] is 7     Bin: 7

Process(8): Number [8] is 8     Bin: 8

Process(9): Number [9] is 9     Bin: 9
```

```
No Of Items in Bin (0) is 1
No Of Items in Bin (1) is 1
No Of Items in Bin (2) is 1
No Of Items in Bin (3) is 1
No Of Items in Bin (4) is 1
No Of Items in Bin (5) is 1
No Of Items in Bin (6) is 1
No Of Items in Bin (7) is 1
No Of Items in Bin (8) is 1
No Of Items in Bin (9) is 6
```

Name :- Nisarg Amlani
Roll :- CE001
Id :- 22ceueg082

LAB-12

1) To the header.h file created in LAB 10 include barrier_init() and barrier() function.

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <errno.h>
int process_fork(int nproc)
{
int j;
for (int j = 1; j < nproc; j++)
   {
       if (fork() == 0)
           return j;
   }
return 0;
}

void process_join(int nproc, int id)
{
int i;
if (id == 0)
   {
       for (i = 1; i < nproc; i++)
           wait(0);
   }
else
exit(0);
```

```
{
   *shmid = shmget(IPC_PRIVATE, size, 0666 | IPC_CREAT);
   if (*shmid == -1)
   {
       perror("shmget failed");
       exit(1);
   }
   char *shm_addr = (char *)shmat(*shmid, 0, 0);
   if (shm_addr == (char *)-1)
   {
       perror("shmat failed");
       exit(1);
   }
   return shm_addr;
}

void spin_lock_init(int *lock, int *condition)

{
   int control;
   *lock = semget(IPC_PRIVATE, 1, 0666 | IPC_CREAT);
   if (*condition == 1)
       control = 0;
   else
       control = 1;

   semctl(*lock, 0, SETVAL, control);
}
void spin_lock(int *lock)
{
   struct sembuf operations;
   operations.sem_num = 0;
   operations.sem_op = -1;
   operations.sem_flg = 0;
   semop(*lock, &operations, 1);
}
void spin_unlock(int *lock)
{
   struct sembuf operations;
   operations.sem_num = 0;
```

```
    operations.sem_op = 1;
    operations.sem_flg = 0;

    semop(*lock, &operations, 1);
}
void barrier_init(int *bar1, int bnum)

{
    int c = 0;
    *(bar1 + 0) = bnum;
    *(bar1 + 1) = 0;
    *(bar1 + 2) = 0;
    spin_lock_init((bar1 + 3), &c);
}
void barrier(int *bar)
{
    int incr = 0;
    while (1)
    {
        spin_lock((bar + 3));
        if (incr == 0 && *(bar + 2) > 0)
        {
            spin_unlock((bar + 3));
            continue;
        }
        if (incr == 0)
        {
            *(bar + 1) = *(bar + 1) + 1;
            incr = 1;
        }
        if (*(bar + 1) < *(bar + 0) && (*(bar + 2) == 0))
        {
            spin_unlock((bar + 3));
            continue;
        }
        else // release phase
        {
            if (*(bar + 2) == 0)
            // release first process
            {
```

```
                *(bar + 2) = *(bar + 0) - 1;
                *(bar + 1) = 0;

                spin_unlock((bar + 3));
                return;
            }
            else // release rest of processes
            {
                *(bar + 2) = *(bar + 2) - 1;
                spin_unlock((bar + 3));
                return;
            }
        }
        spin_unlock((bar + 3));
        printf("\n Error in barrier");
        return;
    }
}
```

2)WAP to Calculate standard deviation with help of barrier

```
#include "header.h"
#include <math.h>

#define nproc 4

int main()
{
   float a[20];
   for (int i=0; i<20; i++)
       a[i] = i;

     int shmid, *lock, *bar;
   lock = (int *)shared(sizeof(int), &shmid);
   bar = (int *)shared(sizeof(int) * 4, &shmid);
   spin_lock_init(lock, &(int){0});
   barrier_init(bar, nproc);
   float *avg, *deviation, sum = 0.0, sum_sq = 0.0;
   deviation = (float *)shared(sizeof(float), &shmid);
   avg = (float *)shared(sizeof(float), &shmid);
   *avg = 0.0;
   *deviation = 0.0;

   int id = process_fork(nproc);

   for (int i=id; i<20; i+=nproc){
       sum = sum + a[i];
   }
   spin_lock(lock);
       *avg = *avg + sum/20;
   spin_unlock(lock);

   barrier(bar);

   sum = 0.0;
   for(int i=id; i<20; i+=nproc)
       sum_sq += pow(a[i] - *avg, 2);
```

```
spin_lock(lock);
        *deviation += sum_sq;
    spin_unlock(lock);

process_join(nproc, id);
printf("all processes joined . \n");
*deviation = *deviation / 20;
printf("Deviation = %f\n", *deviation);
printf("Standard Deviation = %f\n", sqrt(*deviation));

return 0;
}
```

Output:

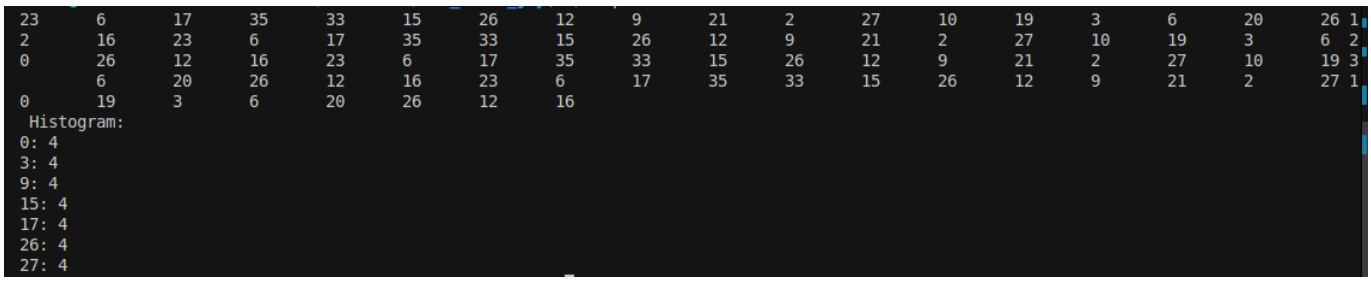
```
all processes joined .
Deviation = 33.250000
Standard Deviation = 5.766281
```

3)Write a parallel program to implement Histogram with barrier.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <unistd.h>
#include "header.h"
#define MAX_VALUE 40
#define NPROC 4

int main()
{
int shmid;
int *histogram;
int *barrier_data; int
   data_size = 20;
int *data = (int *)malloc(data_size * sizeof(int));

for (int i = 0; i < data_size; i++)
   {
data[i] = rand() % MAX_VALUE; printf("%d
       \t",data[i]);
   }

histogram = (int *)shared(MAX_VALUE * sizeof(int), &shmid); barrier_data
   = (int *)shared(4 * sizeof(int), &shmid);
memset(histogram, 0, MAX_VALUE * sizeof(int)); barrier_init(barrier_data,
   NPROC);

int id = process_fork(NPROC);
int local_histogram[MAX_VALUE] = {0};
for (int i = id-1; i < data_size; i+=(NPROC-1) )
   {
local_histogram[data[i]]++;
```

```
    barrier(barrier_data);

    if (id == 0)
    {
        for (int i = 0; i < NPROC; i++)
        {
            for (int j = 0; j < MAX_VALUE; j++)
            {
                histogram[j] += local_histogram[j];
            }
        }
    }
    process_join(NPROC, id);
    if (id == 0)
    {
        printf("\n Histogram:\n");
        for (int i = 0; i < MAX_VALUE; i++)
        {
            if (histogram[i] > 0)
            {
                printf("%d: %d\n", i, histogram[i]);
            }
        }
    }


    shmdt(histogram);
    shmctl(shmid, IPC_RMID, NULL);

    free(data);
    return 0;
}
```

Output:


23 6 17 35 33 15 26 12 9 21 2 27 10 19 3 6 20 26 1
2 16 23 6 17 35 33 15 26 12 9 21 2 27 10 19 3 6 2
0 26 12 16 23 6 17 35 33 15 26 12 9 21 2 27 10 19 3
6 20 26 12 16 23 6 17 35 33 15 26 12 9 21 2 27 1
0 19 3 6 20 26 12 16
Histogram:
0: 4
3: 4
9: 4
15: 4
17: 4
26: 4
27: 4