Name Priyadarshi Prabhakar SAP ID 590029237

EXPERIMENT 4 : VARIABLES AND SCOPE OF VARIABLES

Activity 1: *Declare a global variable outside all functions and use it inside various functions to understand its accessibility.*

ALGORITHM

STEP 1: Start the program.
STEP 2: Declare and initialize a global variable num = 5
STEP 3: Define the function show()
STEP 4: In show(), print the current value of num.
STEP 5: Define the function change()
STEP 6: In change(), assign num = 10 and print new value of num
STEP 7: In main(), print the initial value of num
STEP 8: Call the function show()
STEP 9: Call the function change()
STEP 10: Print value of num in main()
STEP 11: End

PSEUDOCODE :

```
START
```

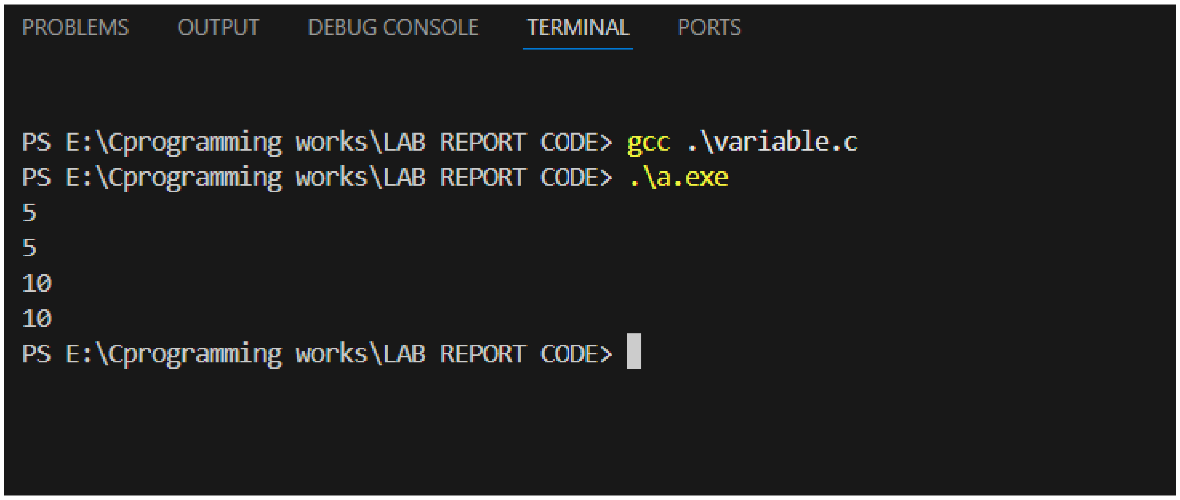
```
DECLARE num = 5

FUNCTION show()
    PRINT num
END FUNCTION

FUNCTION change()
    SET num = 10
    PRINT num
END FUNCTION

MAIN
    PRINT num
    CALL show()
    CALL change()
    PRINT num
END MAIN
END
```

CODE :

```
#include <stdio.h>

int num = 5;
```

```
void show() {
    printf("%d\n", num);
}

void change() {
    num = 10;
    printf("%d\n", num);
}

int main() {
    printf("%d\n", num);
    show();
    change();
    printf("%d\n", num);
    return 0;
}
```

OUTPUT:



```
PS E:\Cprogramming works\LAB REPORT CODE> gcc .\variable.c
PS E:\Cprogramming works\LAB REPORT CODE> .\a.exe
5
5
10
10
PS E:\Cprogramming works\LAB REPORT CODE>
```

Activity 2: *Declare a local variable inside a function and try to access it outside the function. Compare this with accessing the global variable from within the function.*

ALGORITHM :

STEP 1: Start
STEP 2: Declare and initialize a global variable *globalvar*
STEP 3: Define the function localexample()
STEP 4: Declare a local variable *localvar* inside localexample() and initialize it.
STEP 5: Inside localexample(), print localvar and globalvar.
STEP 6: In main(), print the value of globalvar.
STEP 7: Call the function localexample().
STEP 9: End

PSEUDOCODE:

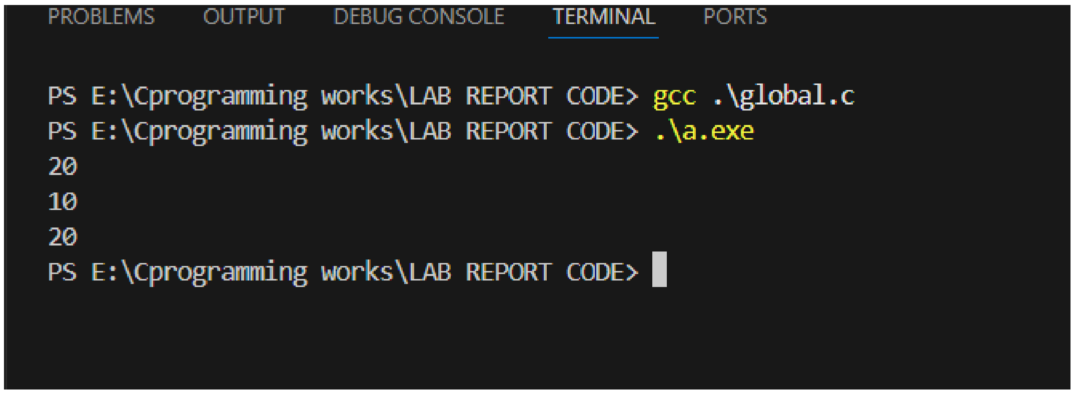
```
START

DECLARE globalvar


FUNCTION localexample()

   DECLARE localvar

   PRINT localvar

   PRINT globalvar

END FUNCTION


MAIN

   PRINT globalvar

   CALL localexample()

END MAIN

END
```

CODE :

```
#include <stdio.h>


int globalvar = 20;
```

```
void localexample() {
    int localvar = 10;
    printf("%d\n", localvar);
    printf("%d\n", globalvar);
}

int main() {
    printf("%d\n", globalvar);
    localexample();
    return 0;
}
```

OUTPUT :



```
PS E:\Cprogramming works\LAB REPORT CODE> gcc .\global.c
PS E:\Cprogramming works\LAB REPORT CODE> .\a.exe
20
10
20
PS E:\Cprogramming works\LAB REPORT CODE>
```

Activity 3: *Declare variables within different code blocks and (enclosed by curly braces) and test their accessibility within and outside those blocks.*

ALGORITHM :

STEP 1: Start

STEP 2: Declare and initialize variable x in the main()

STEP 3: Print the value of x in the main block

STEP 4: Enter inner block 1

STEP 5: Declare and initialize variable y in inner block 1

STEP 6: Print the values of x and y inside inner block 1

STEP 7: Enter inner block 2 inside inner block 1

STEP 8: Declare and initialize variable z in inner block 2

STEP 9: Print the values of x, y, and z inside inner block 2

STEP 10: Exit inner block 2

STEP 11: Exit inner block 1

STEP 12: End

PSEUDOCODE :

```
#include <stdio.h>


int main() {
    int x = 5;
    printf("%d\n", x);


    {
        int y = 10;
        printf("%d %d\n", x, y);


        {
            int z = 15;
            printf("%d %d %d\n", x, y, z);
        }
    }

    return 0;
}
```

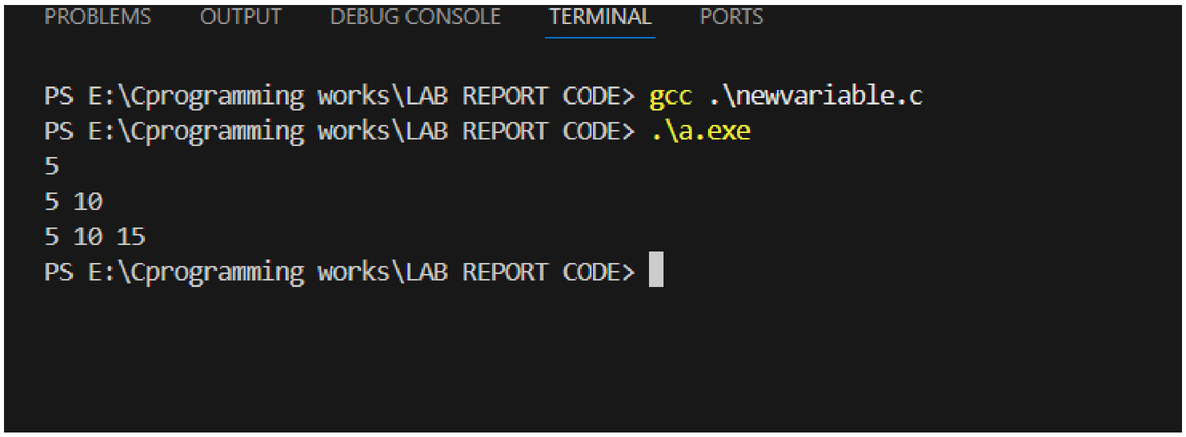
CODE :

```
#include <stdio.h>

int main() {
    int x = 5;
    printf("%d\n", x);

    {
        int y = 10;
        printf("%d %d\n", x, y);

        {
            int z = 15;
            printf("%d %d %d\n", x, y, z);
        }
    }
```

return 0;

}

OUTPUT :



```
PS E:\Cprogramming works\LAB REPORT CODE> gcc .\newvariable.c
PS E:\Cprogramming works\LAB REPORT CODE> .\a.exe
5
5 10
5 10 15
PS E:\Cprogramming works\LAB REPORT CODE>
```

Activity 4: *Declare a static local variable inside a function. Observe how its value persists across function calls.*

ALGORITHM :

STEP 1: Start

STEP 2: Define the function counter() with static local variable count=0

STEP 3: Increment count by 1

STEP 4: Print the value of count

STEP 5: In main(), call counter() for the first time

STEP 6: Call counter() for the second time

STEP 7: Call counter() for the third time

STEP 8: End

PSEUDOCODE :

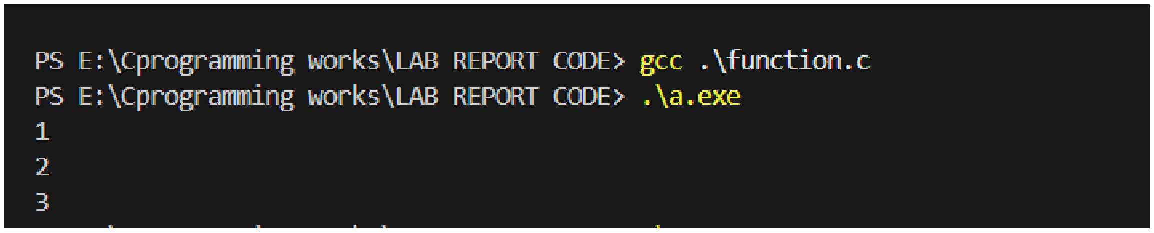
```
START

FUNCTION counter()
   STATIC count = 0
   INCREMENT count
   PRINT count
END FUNCTION

MAIN
   CALL counter()
   CALL counter()
   CALL counter()
END MAIN

END
```

CODE :

```
#include <stdio.h>

void counter() {
   static int count = 0;
```

```
    count++;

    printf("%d\n", count);

}


int main() {

    counter();

    counter();

    counter();

    return 0;

}
```

OUTPUT :



```
PS E:\Cprogramming works\LAB REPORT CODE> gcc .\function.c
PS E:\Cprogramming works\LAB REPORT CODE> .\a.exe
1
2
3
```

Static variable count retained its value between function calls, unlike normal local variables which are reinitialized each time.