1. **Rewrite the following code using a do-while statement with no decisions in the loop body:**

sum = 0;

for (odd = 1; odd < n; odd = odd + 2)

sum = sum + odd;

printf("Sum of the positive odd numbers less than %d is %d\n", n, sum);

sum = 0;

odd = 1;

do {

sum = sum+odd;

odd = odd+2;

} while (odd < n);

printf("Sum of the positive odd numbers less than %d is %d\n", n, sum);

1. **Write nests of loops that cause the following output to be displayed:**

0

0 1

0 1 2

0 1 2 3

0 1 2 3 4

0 1 2 3 4 5

0 1 2 3 4

0 1 2 3

0 1 2

0 1

0

Answer:

Given the relative complexity of the task, design first a solution to the problem before starting your code

The pattern has two sides

0

Increasing depth

0 1

0 1 2

0 1 2 3

0 1 2 3 4

0 1 2 3 4 5

0 1 2 3 4

0 1 2 3

Decreasing depth

0 1 2

0 1

0

If we take the rightmost edge (edge)5 as a reference, we can note that:

* The top side has “edge+1” lines. In each line we print a sequence of integer values. If we generate this side content using a loop, with iterator i that’s incremented in every iteration, we can note that the sequence to print in every line is [0:i]

Iterator i values

0 0

1. 0 1
2. 0 1 2
3. 0 1 2 3
4. 0 1 2 3 4
5. 0 1 2 3 4 5 where 5 corresponds to edge

* The bottom side has “edge” lines. The depth of each line decreases at every iteration. If we generate this side’s content using a loop, with iterator i that’s decremented in every iteration, we can note that the sequence to print in every iteration is [0:i]:

Iterator i value

1. 0 1 2 3 4 , where 4 corresponds to “edge-1”
2. 0 1 2 3
3. 0 1 2
4. 0 1

0 0

Given our design solution above we can deduce the following code:

#include<stdio.h>

#define EDGE 5

int main() {

int i,j;

for (i=0; i<=EDGE;i++)

{

for (j=0; j<=i;j++)

printf("%d ", j);

printf("\n");

}

for (i=EDGE-1; i>=0;i--)

{

for (j=0; j<=i;j++)

printf("%d ", j);

puts("");

}

return 0;

}

Try it with different EDGE value in the define directive.

IMPORTANT: please note that we need to add { after the first outer loop; otherwise printf(“\n) or puts will be out of the outer loop’s scope; try it to see the effect.

and separated by spaces.

1. **Write a do-while loop that repeatedly prompts for and takes input until a value in the range 0 through 15 inclusive is input. Include code that prevents the loop from executing forever on input of a wrong data type.**

Answer:

If we did not have to verify that the read data is of its specified data type, the following code will be sufficient:

#include<stdio.h>

#include<stdbool.h>

int main(void)

{

int data;

bool repeat= false;

do

{

if(repeat)

printf("\n invalid value, your input should be integer in the range [0:15]: ");

else {

printf("\n please input an integer value in [0:15]: ");

repeat=true;

}

scanf("%d", &data);

} while (data <0 || data> 15);

printf(" Valid Data: %d", data);

}

The above code works fine, if the input data is always an integer. If it is not, and the local variable data is not initialized to zero (being a local variable, there is no guarantee its value would be initialized to zero), the above code leads to infinite loop (try it!) as scanf will keep trying to read the data from the same (non-empty) input buffer.

To ascertain that the read value by scanf was indeed an integer, we need to check the return value of scanf as follows:

status=scanf(“%d”, &data);

The value of *status* will be 1 if scanf read one integer value as instructed by the format specifier %d. This is valid for any format specifier regardless even of their numbers in the scanf argument.

If the input value is not integer (i.e. status!=1), we need to clear the buffer first from the user input before reading a new , hopefully valid, data from the user. To clear the keyboard buffer, we have to read its content.

To this end, we first need to define an array of characters to store the input’s line:

#define LINE\_SIZE 100

char line[LINE\_SIZE];

Then use the regular expression [^\n] to read all characters up to new line: scanf(“%[^\n]”, line)

(I might have used %[^\n]s, which is correct in most of the cases)

The function definition of the above solution is:

void skip\_line\_format(){

char line [LINE\_SIZE];

scanf("%[^\n]",line);

return;

}

IF NEEDED , THE CODE OF THE ABOVE FUNCTION WILL BE PROVIDED.

Because line is an array, we don’t need to prefix it with “&” in the scanf

The resulting program is:

#include<stdio.h>

#include <stdbool.h>

#define LINE\_SIZE 100

void skip\_line\_char();

int main(void){

bool inValid=true; // while loop conditional expression

int status; // return value of scanf

int data;

/\* Get data from user until in\_val is in the range 0 15. \*/

do {

/\* Get a number from the user. \*/

printf("Enter an integer in the range [0:15]>: ");

status = scanf("%d", &data);

/\* Validate the number. \*/

if (status != 1) { /\* an integer value was NOT read \*/

skip\_line\_format();

printf("InValid Input \n");

}

else if (data < 0 || data > 15) { /\* data is valid but out of range \*/

printf("Number %d is not in requested range.\n", data);

}

else

{

inValid=false; /\* to escale the loop \*/

printf ("Valid Input \n");

}

} while (inValid);

}

void skip\_line\_format(){

char line [LINE\_SIZE];

scanf("%[^\n]",line);

return;

}