Example: a + b && b / c % d \* e || 5 <= 0 != a + b – d

a = 5, b = 6, c = 8, d = 2, and e = 1

A library charges fine for every book returned late. For the first 5 days the fine is Re 1/-, for 6-10 days the fine is Rs 2 and for above 10 days the fine is Rs 5. Translate the above task in to a program in c.

Design a C module that checks whether a triangle can be formed or not considering the angles as arguments. (Hint: In a triangle, the sum of all angles must be 180 degrees)

A company pays car allowance to its employees based on the size of engine in the car. Those employees whose car engines are less than or equal to 1500CC receive Rs 2/- per km and those whose car engines are greater than 1500CC receive an additional 50 paise per km. Develop a module that accepts the engine size and number of kilometers, calculates the car allowance and returns the result.

Given the values of first term (a), and the common difference (d), generate first n terms in an Arithmetic Progression (AP). Design a modularized code for the specified task. (Hint: The first n terms in AP are a, a+d, a+2d, a+3d, .. a+(n-1)d)

Develop a function that accepts an integer argument and prints the digits of n that divide n. For example, if n is 122 then print 2,2,1 as 2 and 1 divide the number n. If none of the digits divide the number, the print “No factors”.

Storage Classes in C

Storage classes in C are used to determine the lifetime, visibility, memory location, and initial value of a variable. There are four types of storage classes in C

* Automatic
* External
* Static
* Register

Graphical user interface, text, application, email

Description automatically generated

Automatic

* Automatic variables are allocated memory automatically at runtime.
* The visibility of the automatic variables is limited to the block in which they are defined.

The scope of the automatic variables is limited to the block in which they are defined.

* The automatic variables are initialized to garbage by default.
* The memory assigned to automatic variables gets freed upon exiting from the block.
* The keyword used for defining automatic variables is auto.
* Every local variable is automatic in C by default.

#### Example 1

1. #include <stdio.h>
2. int main()
3. {
4. int a; //auto
5. char b;
6. float c;
7. printf("%d %c %f",a,b,c); // printing initial default value of automatic variables a, b, and c.
8. return 0;
9. }

#### ­­Example 2

1. #include <stdio.h>
2. int main()
3. {
4. int a = 10,i;
5. printf("%d ",++a);
6. {
7. int a = 20;
8. for (i=0;i<3;i++)
9. {
10. printf("%d ",a); // 20 will be printed 3 times since it is the local value of a
11. }
12. }
13. printf("%d ",a); // 11 will be printed since the scope of a = 20 is ended.
14. }

Example 3:

#include <stdio.h>

int main( )

{

auto int j = 1;

{

auto int j= 2;

{

auto int j = 3;

printf ( " %d ", j);

}

printf ( "\t %d ",j);

}

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

}

## Static

* The variables defined as static specifier can hold their value between the multiple function calls.
* Static local variables are visible only to the function or the block in which they are defined.
* A same static variable can be declared many times but can be assigned at only one time.
* Default initial value of the static integral variable is 0 otherwise null.
* The visibility of the static global variable is limited to the file in which it has declared.
* The keyword used to define static variable is static.

Example-1

1. #include<stdio.h>
2. static char c;
3. static int i;
4. static float f;
5. static char s[100];
6. void main ()
7. {
8. printf("%c%d%f %s",c,i,f,s); // the initial default value of c, i, and f will be printed.
9. }

Example 2

1. #include<stdio.h>
2. void sum()
3. {
4. static int a = 10;
5. static int b = 24;
6. printf("%d %d \n",a,b);
7. a++;
8. b++;
9. }
10. void main()
11. {
12. int i;
13. sum();
14. sum();
15. sum();

1. }

Example:3

#include <stdio.h> /\* function declaration \*/

void next(void);

static int counter = 7; /\* global variable \*/

main() {

while(counter<10) {

next();

counter++; }

return 0;}

void next( void ) { /\* function definition \*/

static int iteration = 13; /\* local static variable \*/

iteration ++;

printf("iteration=%d and counter= %d\n", iteration, counter);

}

## Register

* The variables defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in the CPU.
* We can not dereference the register variables, i.e., we can not use &operator for the register variable.
* The access time of the register variables is faster than the automatic variables.
* The initial default value of the register local variables is 0.
* The register keyword is used for the variable which should be stored in the CPU register. However, it is compiler?s choice whether or not; the variables can be stored in the register.
* We can store pointers into the register, i.e., a register can store the address of a variable.
* Static variables can not be stored into the register since we can not use more than one storage specifier for the same variable.

Example-1

1. #include <stdio.h>
2. int main()
3. {
4. register int a; // variable a is allocated memory in the CPU register.
5. Scanf(“%d”,&a);
6. printf("%d",a);
7. }

Example-2

1. #include <stdio.h>
2. int main()
3. {
4. register int a = 0;
5. printf("%u",&a); // This will give a compile time error since we can not access the address of a register variable.
6. }

## External

* The external storage class is used to tell the compiler that the variable defined as extern is declared with an external linkage elsewhere in the program.
* The variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in the program.
* The default initial value of external integral type is 0 otherwise null.
* We can only initialize the extern variable globally, i.e., we can not initialize the external variable within any block or method.
* An external variable can be declared many times but can be initialized at only once.
* If a variable is declared as external then the compiler searches for that variable to be initialized somewhere in the program which may be extern or static. If it is not, then the compiler will show an error.

Example

#include <stdio.h>

int count;

void printCount();

int main()

{

count=5;

printCount();

return 0;

}

#include<stdio.h>

extern int count;

void printCount()

{

printf("%d",count); //5

}