Explain data types of C languages with suitable examples. Give memory requirement and range of each also.

In the C programming language, data types are used to define the type of data that a variable can hold. Here are some of the basic data types in C, along with examples, memory requirements, and ranges:

1. **int (Integer):**
    - Example: int num = 42;
    - Memory Requirement: 4 bytes (on most systems)
    - Range: -2,147,483,648 to 2,147,483,647
2. **float (Floating-point):**
    - Example: float pi = 3.14;
    - Memory Requirement: 4 bytes
    - Range: 3.4E +/- 38 (7 digits of precision)
3. **double (Double precision floating-point):**
    - Example: double bigNum = 123.456;
    - Memory Requirement: 8 bytes
    - Range: 1.7E +/- 308 (15 digits of precision)
4. **char (Character):**
    - Example: char grade = 'A';
    - Memory Requirement: 1 byte
    - Range: -128 to 127 or 0 to 255 (depending on signed or unsigned)
5. **short (Short integer):**
    - Example: short smallNum = 32767;
    - Memory Requirement: 2 bytes
    - Range: -32,768 to 32,767
6. **long (Long integer):**
    - Example: long bigInteger = 1234567890L;
    - Memory Requirement: 4 bytes or 8 bytes (system-dependent)
    - Range: -2,147,483,648 to 2,147,483,647 or -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (depending on the system)
7. **long long (Long long integer):**
    - Example: long long hugeInteger = 123456789012345LL;
    - Memory Requirement: 8 bytes
    - Range: -(2^63) to (2^63)-1
8. **unsigned int (Unsigned integer):**

- Example: unsigned int positiveNum = 100;
- Memory Requirement: 4 bytes
- Range: 0 to 4,294,967,295

These are some of the basic data types in C, and their memory requirements and ranges can vary based on the system and compiler. It's important to consider the range and memory usage when choosing a data type for a particular variable to avoid overflow or underflow issues.

Program:

```
#include<stdio.h>

#include<conio.h>

void main(){

        clrscr();

        int a=10;

        float b=10.1;

        char c=’a’;

        printf(“\n%d”, a);

        printf(“\n%d”, b);

        printf(“\n%d”, c);

        getch();

}
```

Output:

```
10

10.1

a
```

Explain various control statements of C language with suitable examples and programs

In C, control statements are used to control the flow of execution in a program. They include conditional statements (if, else if, else), looping statements (for, while, do-while), and branching statements (break, continue, goto). Let's explore each of these with examples:

C if else Statement

The if-else statement in C is used to perform the operations based on some specific condition. The operations specified in if block are executed if and only if the given condition is true.

There are the following variants of if statement in C language.

- If statement
- If-else statement
- If else-if ladder
- Nested if

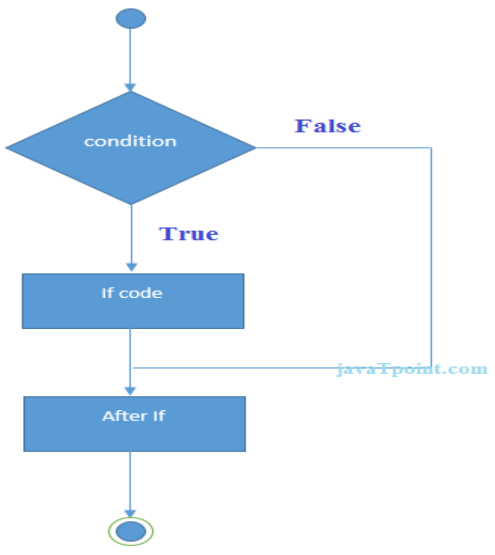
If Statement

The if statement is used to check some given condition and perform some operations depending upon the correctness of that condition. It is mostly used in the scenario where we need to perform the different operations for the different conditions.

The syntax of the if statement is given below.

```
if(expression){

//code to be executed

}
```

Flowchart of if statement in C



Let's see a simple example of C language if statement.

```
#include<stdio.h>

int main(){

int number=0;

printf("Enter a number:");

scanf("%d",&number);

if(number%2==0){

printf("%d is even number",number);

}

return 0;

}
```

Output

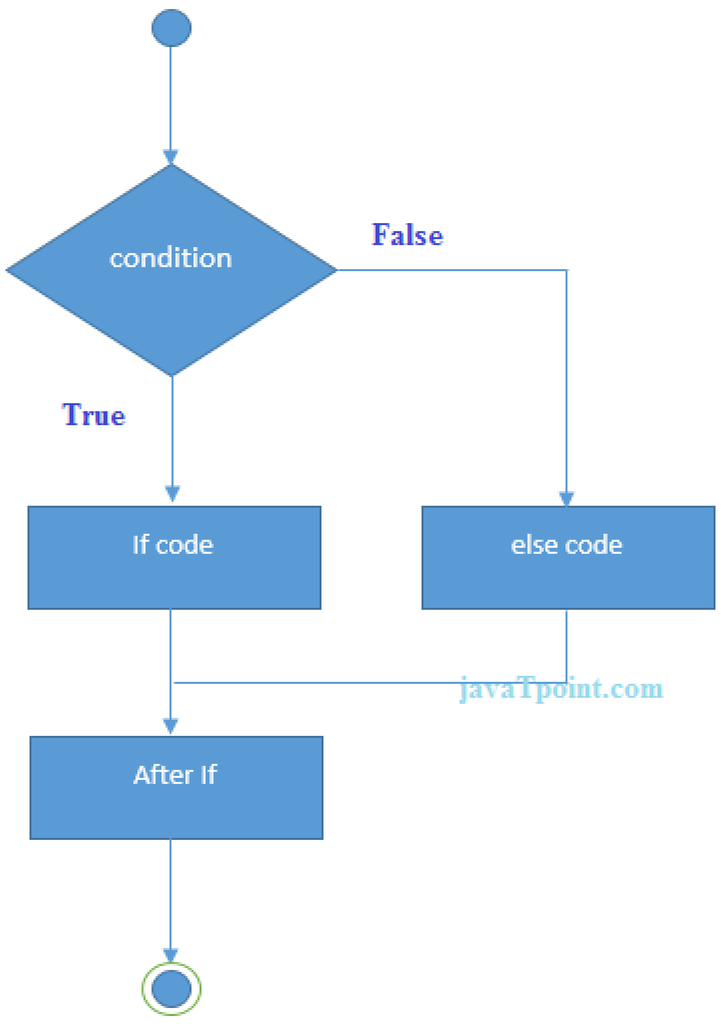
```
Enter a number:4
4 is even number
enter a number:5
```

If-else Statement

The if-else statement is used to perform two operations for a single condition. The if-else statement is an extension to the if statement using which, we can perform two different operations, i.e., one is for the correctness of that condition, and the other is for the incorrectness of the condition. Here, we must notice that if and else block cannot be executed simiulteneously. Using if-else statement is always preferable since it always invokes an otherwise case with every if condition. The syntax of the if-else statement is given below.

```
if(expression){

//code to be executed if condition is true

}else{

//code to be executed if condition is false

}
```

Flowchart of the if-else statement in C



Let's see the simple example to check whether a number is even or odd using if-else statement in C language.

```
#include<stdio.h>

int main(){

int number=0;

printf("enter a number:");

scanf("%d",&number);

if(number%2==0){

printf("%d is even number",number);

}

else{

printf("%d is odd number",number);
}

return 0;

}
```

Output

```
enter a number:4
4 is even number
enter a number:5
5 is odd number
```

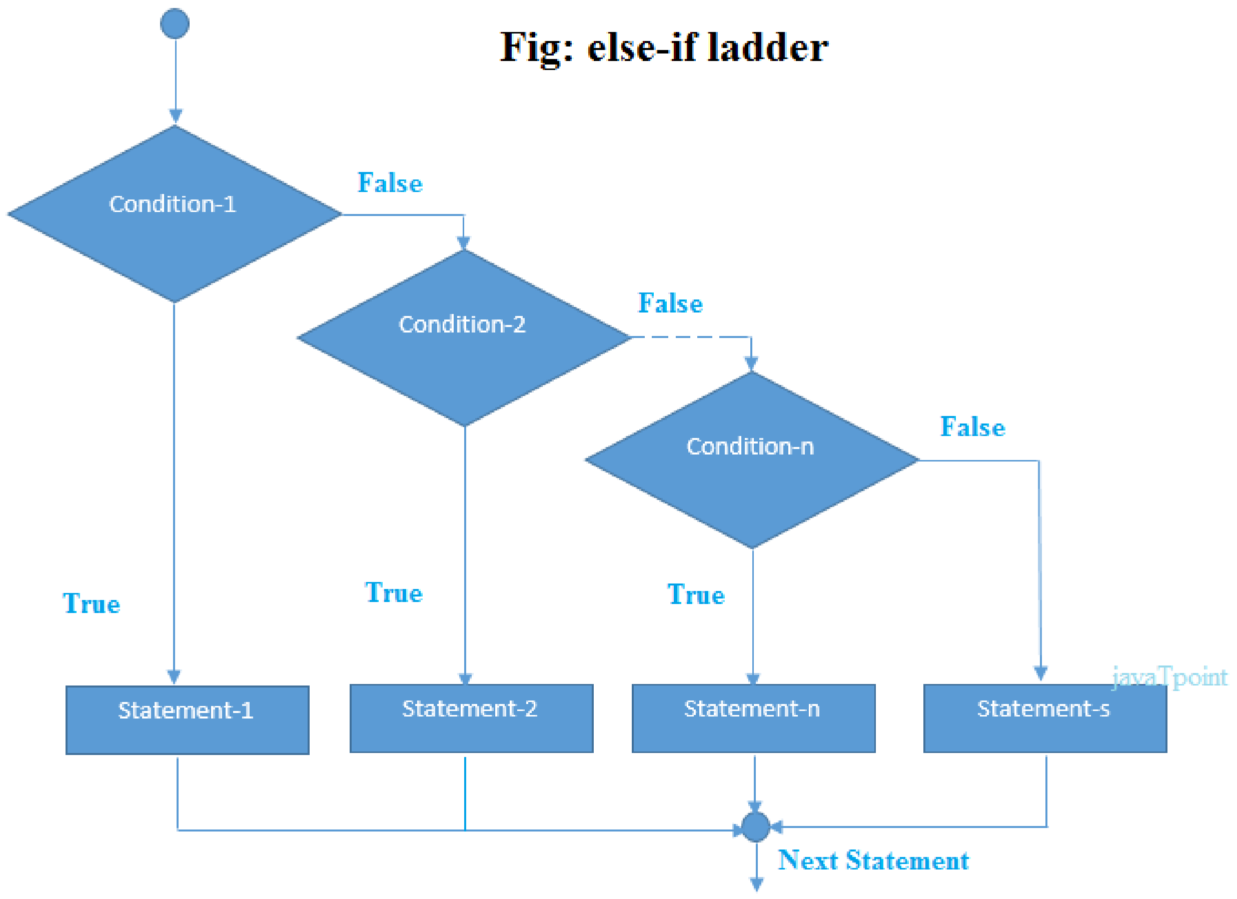
If else-if ladder Statement

The if-else-if ladder statement is an extension to the if-else statement. It is used in the scenario where there are multiple cases to be performed for different conditions. In if-else-if ladder statement, if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed. There are multiple else-if blocks possible. It is similar to the switch case statement where the default is executed instead of else block if none of the cases is matched.

```
if(condition1){

//code to be executed if condition1 is true

}else if(condition2){

//code to be executed if condition2 is true

}

else if(condition3){

//code to be executed if condition3 is true

}

...

else{

//code to be executed if all the conditions are false

}
```

Flowchart of else-if ladder statement in C

Fig: else-if ladder



The example of an if-else-if statement in C language is given below.

```
#include<stdio.h>

int main(){

int number=0;

printf("enter a number:");

scanf("%d",&number);

if(number==10){

printf("number is equals to 10");

}

else if(number==50){

printf("number is equal to 50");

}

else if(number==100){

printf("number is equal to 100");
}

else{

printf("number is not equal to 10, 50 or 100");

}

return 0;

}
```

Output

```
enter a number:4
number is not equal to 10, 50 or 100
enter a number:50
number is equal to 50
```

Program to calculate the grade of the student according to the specified marks.

```
#include <stdio.h>

int main()

{

   int marks;

   printf("Enter your marks?");

   scanf("%d",&marks);

   if(marks > 85 && marks <= 100)

   {

      printf("Congrats ! you scored grade A ...");

   }

   else if (marks > 60 && marks <= 85)

   {

      printf("You scored grade B + ...");

   }

   else if (marks > 40 && marks <= 60)

   {
    printf("You scored grade B ...");
  }
  else if (marks > 30 && marks <= 40)
  {
    printf("You scored grade C ...");
  }
  else
  {
    printf("Sorry you are fail ...");
  }
}
```

Output

```
Enter your marks?10
Sorry you are fail ...
Enter your marks?40
You scored grade C ...
Enter your marks?90
Congrats ! you scored grade A ...
```

C Switch Statement

The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possibles values of a single variable called switch variable. Here, We can define various statements in the multiple cases for the different values of a single variable.

The syntax of switch statement in c language is given below:

```
switch(expression){
case value1:
//code to be executed;
break;  //optional
case value2:

//code to be executed;

break;  //optional

......



default:

code to be executed if all cases are not matched;

}
```

Rules for switch statement in C language

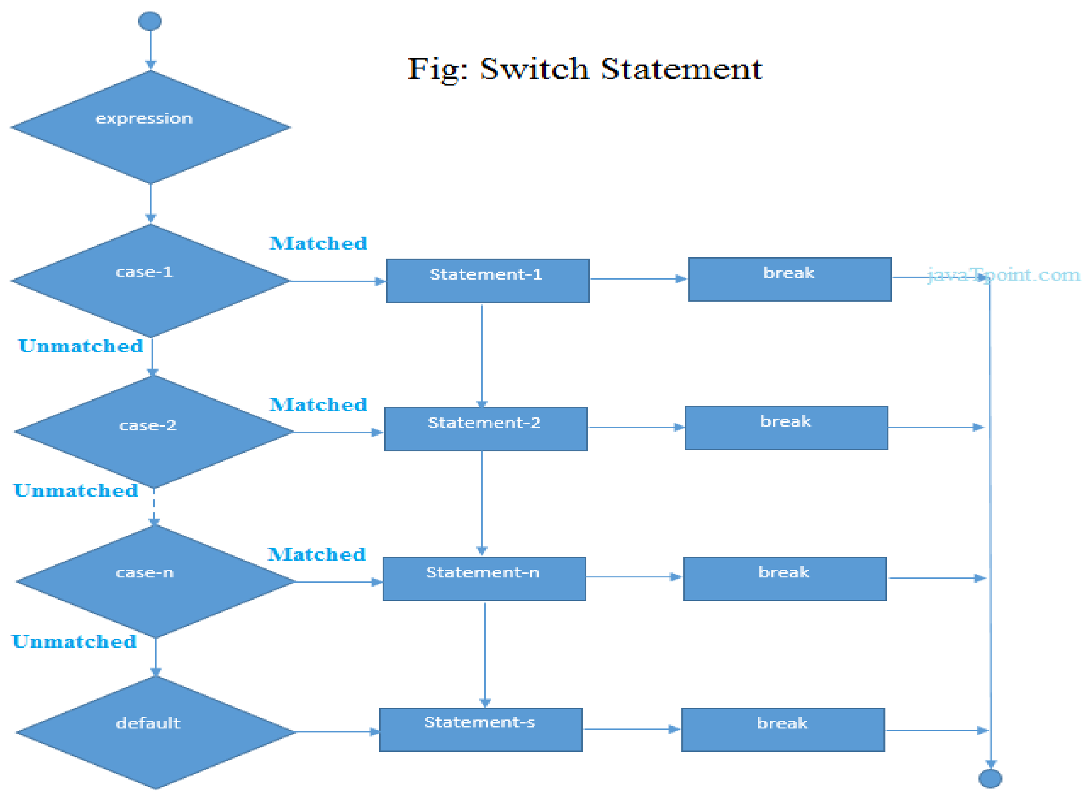
The *switch expression* must be of an integer or character type.

The *case value* must be an integer or character constant.

The *case value* can be used only inside the switch statement.

The *break statement* in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case. It is known as *fall through* the state of C switch statement.

Flowchart of switch statement in C



Functioning of switch case statement

First, the integer expression specified in the switch statement is evaluated. This value is then matched one by one with the constant values given in the different cases. If a match is found, then all the statements specified in that case are executed along with the all the cases present after that case including the default statement. No two cases can have similar values. If the matched case contains a break statement, then all the cases present after that will be skipped, and the control comes out of the switch. Otherwise, all the cases following the matched case will be executed.

Let's see a simple example of c language switch statement.

```
#include<stdio.h>

int main(){

int number=0;

printf("enter a number:");

scanf("%d",&number);
switch(number){

case 10:

printf("number is equals to 10");

break;

case 50:

printf("number is equal to 50");

break;

case 100:

printf("number is equal to 100");

break;

default:

printf("number is not equal to 10, 50 or 100");

}

return 0;

}
```

Output

```
enter a number:4
number is not equal to 10, 50 or 100
enter a number:50
number is equal to 50
```

C Loops

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are three types of loops used in the C language. In this part of the tutorial, we are going to learn all the aspects of C loops.

Why use loops in C language?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the printf statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantage of loops in C

1) It provides code reusability.

2) Using loops, we do not need to write the same code again and again.

3) Using loops, we can traverse over the elements of data structures (array or linked lists).

Types of C Loops

There are three types of loops in C language that is given below:

1. do while
2. while
3. for

do-while loop in C

The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

The syntax of do-while loop in c language is given below:

```
do{

//code to be executed
}while(condition);
```

Program:

```
#include<stdio.h>
#include<conio.h>
void main(){
        int i=1;
        do{
                printf(%d\n”,i);
        }while(i<=5);
        getch();
}
```

Output:

```
1
2
3
4
5
```

while loop in C

The while loop in c is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.

The syntax of while loop in c language is given below:

```
while(condition){

//code to be executed

}
```

Program:

```
#include<stdio.h>

#include<conio.h>

void main(){

    int i=1;

    while(i<=5){

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

    }

    getch();

}
```

Output:

```
1

2

3

4

5
```

for loop in C

The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance.

The syntax of for loop in c language is given below:

```
for(initialization;condition;incr/decr){

//code to be executed

#include<stdio.h>

#include<conio.h>

void main(){

    int i;

    for(i=1;i<=5;i++){

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

    }

    getch();

}
```

Output:

```
1

2

3

4

5
```

Nested Loops in C

C supports nesting of loops in C. **Nesting of loops** is the feature in C that allows the looping of statements inside another loop. Let's observe an example of nesting loops in C.

Any number of loops can be defined inside another loop, i.e., there is no restriction for defining any number of loops. The nesting level can be defined at n times. You can define any type of loop inside another loop; for example, you can define '**while**' loop inside a '**for**' loop.

Syntax of Nested loop

```
Outer_loop

{

   Inner_loop

  {

       // inner loop statements.

  }

     // outer loop statements.

}
```

Outer_loop and **Inner_loop** are the valid loops that can be a 'for' loop, 'while' loop or 'do-while' loop.

Nested for loop

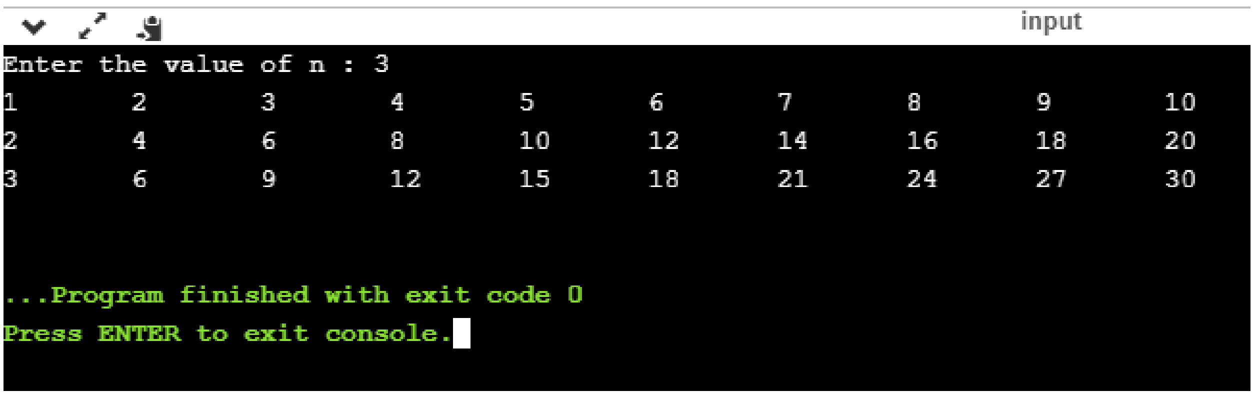
The nested for loop means any type of loop which is defined inside the 'for' loop.

```
for (initialization; condition; update)

{

   for(initialization; condition; update)

   {

        // inner loop statements.

   }

   // outer loop statements.

}
```

Example of nested for loop

```
#include <stdio.h>

int main()

{

  int n;// variable declaration

   printf("Enter the value of n :");

  // Displaying the n tables.

   for(int i=1;i<=n;i++)  // outer loop

  {

       for(int j=1;j<=10;j++)  // inner loop

     {

           printf("%d\t",(i*j)); // printing the value.

     }

       printf("\n");

  }
```

Output:



```
Enter the value of n : 3
1       2       3       4       5       6       7       8       9       10
2       4       6       8       10      12      14      16      18      20
3       6       9       12      15      18      21      24      27      30


...Program finished with exit code 0
Press ENTER to exit console.
```

Nested while loop

The nested while loop means any type of loop which is defined inside the 'while' loop.

```
while(condition)

{

    while(condition)

   {
        // inner loop statements.
   }
// outer loop statements.
}
```

Nested do..while loop

The nested do..while loop means any type of loop which is defined inside the 'do..while' loop.

```
do
{
   do
  {
      // inner loop statements.
   }while(condition);
// outer loop statements.
}while(condition);
```