Computer And Programming Fundamentals With C

unit -2(Control structures and Input/Output functions)

What is if and if-else in C?

C has the conditional statements

<u>if-</u>Use if to specify a block of code to be executed, if a specified condition is true.

The syntax of the if statement in C programming is:

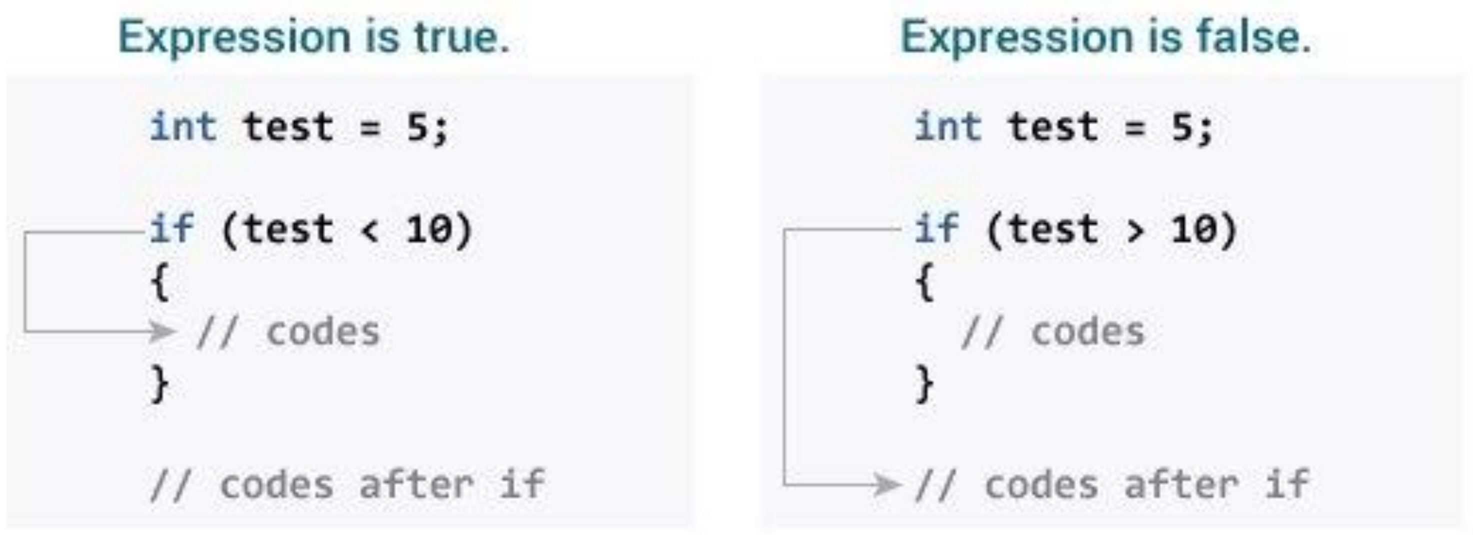
```
if (test expression)
{
  // code
}
```

How if statement works?

The if statement evaluates the test expression inside the parenthesis ().

If the test expression is evaluated to true, statements inside the body of if are executed.

If the test expression is evaluated to false, statements inside the body of if are not executed.



```
int test = 5;

if (test < 10)
{
  // codes
}

// codes after if
```

```
int test = 5;

if (test > 10)
{
  // codes
}

// codes after if
```

C if...else Statement

else-

- The if statement may have an optional else block. The syntax of the if..else statement is:

```
if (test expression)
 {    // run code if test expression is true
 }
else
{    // run code if test expression is false
}
```

How if...else statement works?

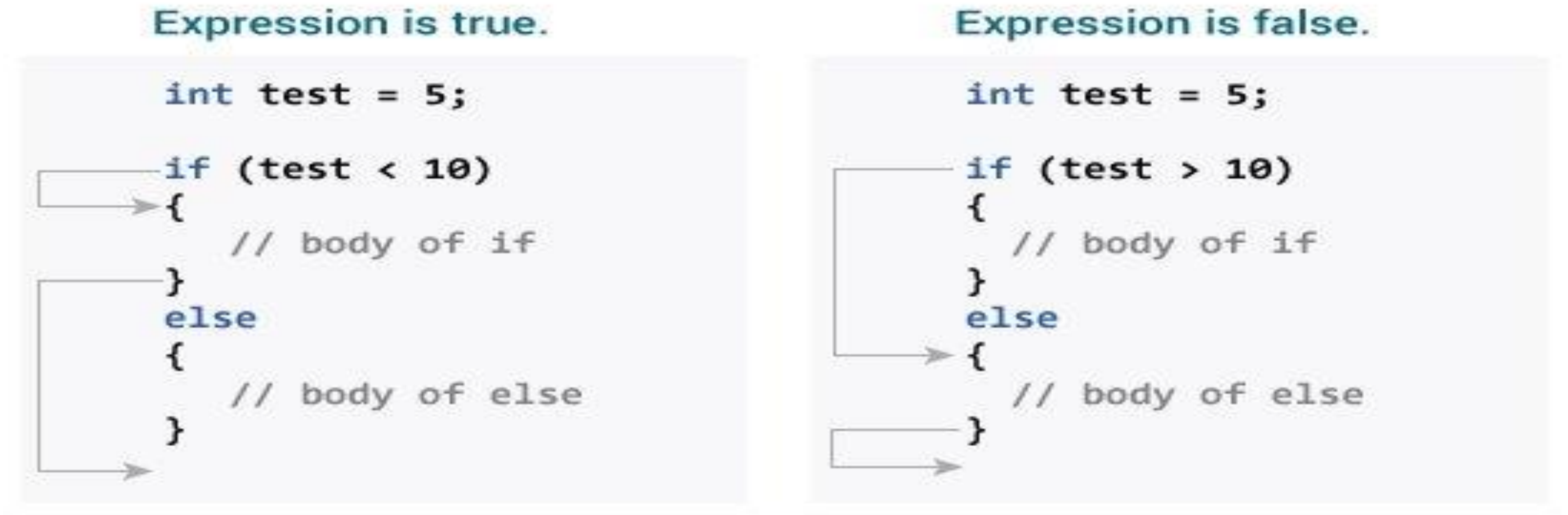
If the test expression is evaluated to true,

•statements inside the body of if are executed.

If the test expression is evaluated to false

•statements inside the body of else are executed

•statements inside the body of if are skipped from execution



```
int test = 5;

if (test < 10)
{
   // body of if
}
else
{
   // body of else
}
```

```
int test = 5;

if (test > 10)
{
   // body of if
}
else
{
   // body of else
}
```

C if...else Ladder

The if...else statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.

The if...else ladder allows you to check between multiple test expressions and execute different statements.

Syntax of if...else Ladder

```
if (test expression1)
 {   // statement(s)
}
else if(test expression2)
{   // statement(s)
}
else if (test expression3)
{   // statement(s)
}..
else
{   // statement(s)
}
```

Example 3: C if...else Ladder

```
#include <stdio.h>
int main()
{
 int number1, number2;
 printf("Enter two integers: ");
 scanf("%d %d", &number1, &number2);
 if(number1 == number2)
   {                                    //checks if the two integers are equal.
  printf("Result: %d = %d",number1,number2);
   }
else if (number1 > number2)
   {                                    //checks if number1 is greater than number2.
 printf("Result: %d > %d", number1, number2);
    }
else
   {                                    //checks if both test expressions are false
  printf("Result: %d < %d",number1, number2);
   }
   return 0;
}
```

Output

```
Enter two integers: 12 23
Result: 12 < 23
```

Nested if...else

It is possible to include an if...else statement inside the body of another if...else statement.

```
#include <stdio.h>
int main() {
int number1, number2;
printf("Enter two integers: ");
scanf("%d %d", &number1, &number2);
if (number1 >= number2)
 {
 if (number1 == number2)
 {
printf("Result: %d = %d",number1,number2);
    }
     else
{
    printf("Result: %d > %d", number1, number2);
  }
    }
    else
 {
      printf("Result: %d < %d",number1, number2);
   }
   return 0;
}
```

:

Loops

- Loops can execute a block of code as long as a specified condition is reached.
- Loops are handy because they save time, reduce errors, and they make code more readable.

C programming has three types of loops.

- for loop
- while loop
- do...while loop

While Loop

The while loop loops through a block of code as long as a specified condition is true:

Syntax

```
while (condition)
{
  // code block to be executed
}
```

How while loop works?

- The while loop evaluates the test Expression inside the parentheses ().
- If test Expression is **true**, statements inside the body of while loop are executed. Then, test Expression is evaluated again.
- The process goes on until test Expression is evaluated to **false**.
- If test Expression is **false**, the loop terminates (ends).

Example

- In the example below, the code in the loop will run, over and over again, as long as a variable (i) is less than 5

```
int i = 0;
while (i < 5)
 {
 printf("%d\n", i);
 i++;
}
```

output:

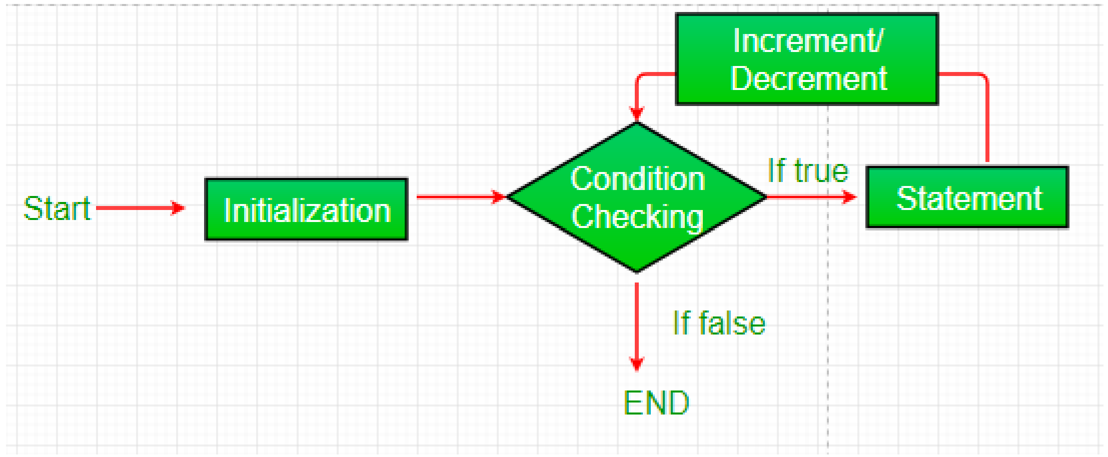
```
0
1
2
3
4
```

For Loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

for loop:
for loop provides a concise way of writing the loop structure. a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.
Syntax:
```
for (initialization condition; testing condition; increment/decrement) { statement(s) }
```

Flowchart:



Example

```
#include<stdio.h>
int main()
{
int i=0;
for(i=1;i<=10;i++)
{
printf("%d \n",i);
}
return 0;
}
```

```
Output:
1
2
3
4
5
6
7
8
9
10
```

The Do/While Loop

The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed at least once. Only then, the test expression is evaluated.

Syntax

```
•do {
  // code block to be executed
}
while (condition);
```

How do...while loop works?

- The body of do...while loop is executed once. Then, the test Expression is evaluated.
- If test Expression is **true**, the body of the loop is executed again and test Expression is evaluated once more.
- This process goes on until test Expression becomes **false**.
- If test Expression is **false**, the loop ends.

Example

The example below uses a do/while loop.The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

```
• int i = 0;

  do {
    printf("%d\n", i);
    i++;
  }
  while (i < 5);
```

Nested Loops

C programming allows to use one loop inside another loop.
The following section shows a few examples to illustrate the concept.

Syntax
The syntax for a **nested for loop**
statement in C is as follows −

```
for ( init; condition; increment )
{
for ( init; condition; increment )
{
 statement(s);
 }
 statement(s);
 }
```

The syntax for a **nested while loop** statement in C programming language is as follows −

```
while(condition)
 {
 while(condition)
 {
 statement(s);
 }
 statement(s);
```

Switch Statement in C

- Instead of writing **many** if..else statements, you can use the switch statement.
- The switch statement selects one of many code blocks to be executed:
- Basically, it is used to perform different actions based on different conditions(cases).
- The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
- In C, the switch case statement is used for executing one condition from multiple conditions. It is similar to an if-else-ladder.
- The switch statement consists of conditional-based cases and a default case.

Rules of the switch case statement

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- The default statement is optional, and specifies some code to run if there is no case match.

Syntax of switch Statement in C

```
switch(expression)
{
case value1: statement_1;
          break;
case value2: statement_2;
           break;
.
.
.
case value_n: statement_n;
               break;
default: default_statement;
}
```

C program to Demonstrate returning of day based numeric

```
#include <stdio.h>
 int main()
{
    int var = 1;
    switch (var) {
        case 1:
            printf("Case 1 is Matched.");
            break;
        case 2:
   printf("Case 2 is Matched.");
            break;
         case 3:
            printf("Case 3 is Matched.");
            break;
        default:
            printf("Default case is Matched.");
           break;
    }
           return 0;
}
```

Output

```
Case 1 is Matched.
```

Break statement

- **What is break in C?**

  The **break in C** is a loop control statement that breaks out of the loop. It can be used inside loops or switch statements to bring the control out of the block. The break statement can only break out of a single loop at a time.

- **The break statement is one of the four jump statements in the C language. The purpose of the break statement in C is for unconditional exit from the loop.**

Syntax of break in C

```
break;
```

We just put the break where ever we want to terminate the execution of the loop.

using break inside while loop

```
printf("\nbreak in while loop\n");

int i = 1;

while (i < 20)

 {

 if (i == 3)

 break;

else

printf("%d ", i);  i++;

}

return 0;}
```

Output

```
break in while loop

1 2
```

continue statements

- The **continue statement in C** is a jump statement that is used to bring the program control to the start of the loop. We can use the continue statement in the while loop, for loop, or do..while loop to alter the normal flow of the program execution. Unlike break, it cannot be used with a C switch case.

Syntax of continue in C

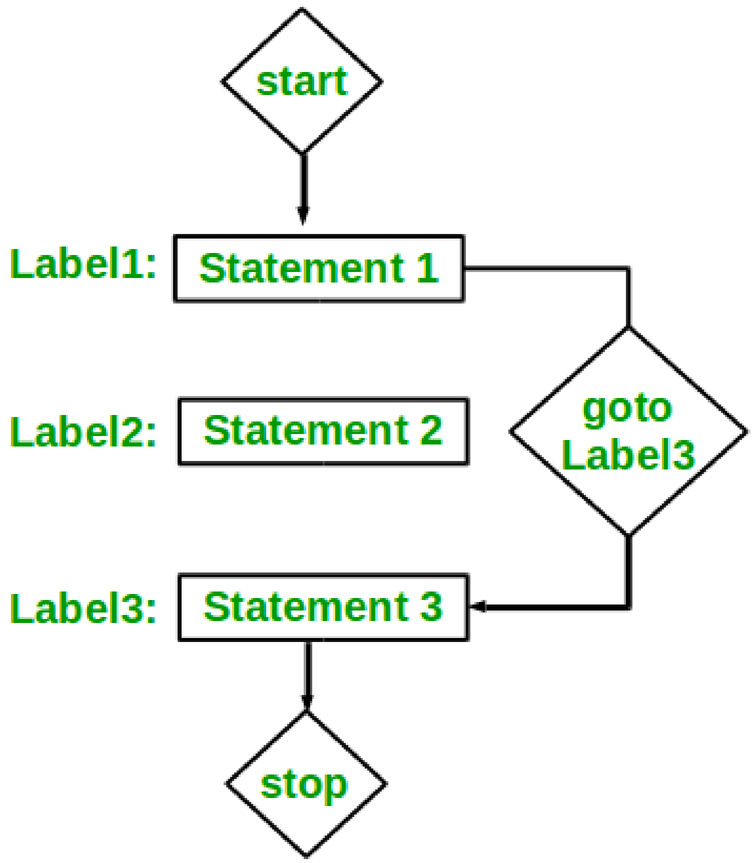
The syntax of continue is just the continue keyword placed wherever we want in the loop body.

```
continue;
```

Goto, Return

- The **C goto statement** is a jump statement which is sometimes also referred to as an **unconditional jump** statement. The goto statement can be used to jump from anywhere to anywhere within a function.
- **Syntax**:
- Syntax1 | Syntax2
- ----------------------------
- goto label; | label:
- . | .
- . | .
- . | .
- label: | goto label;
- In the above syntax, the first line tells the compiler to go to or jump to the statement marked as a label. Here, the label is a user-defined identifier that indicates the target statement. The statement immediately followed after ‘label:’ is the destination statement. The ‘label:’ can also appear before the ‘goto label;’ statement in the above syntax.

Flowchart :Goto statement

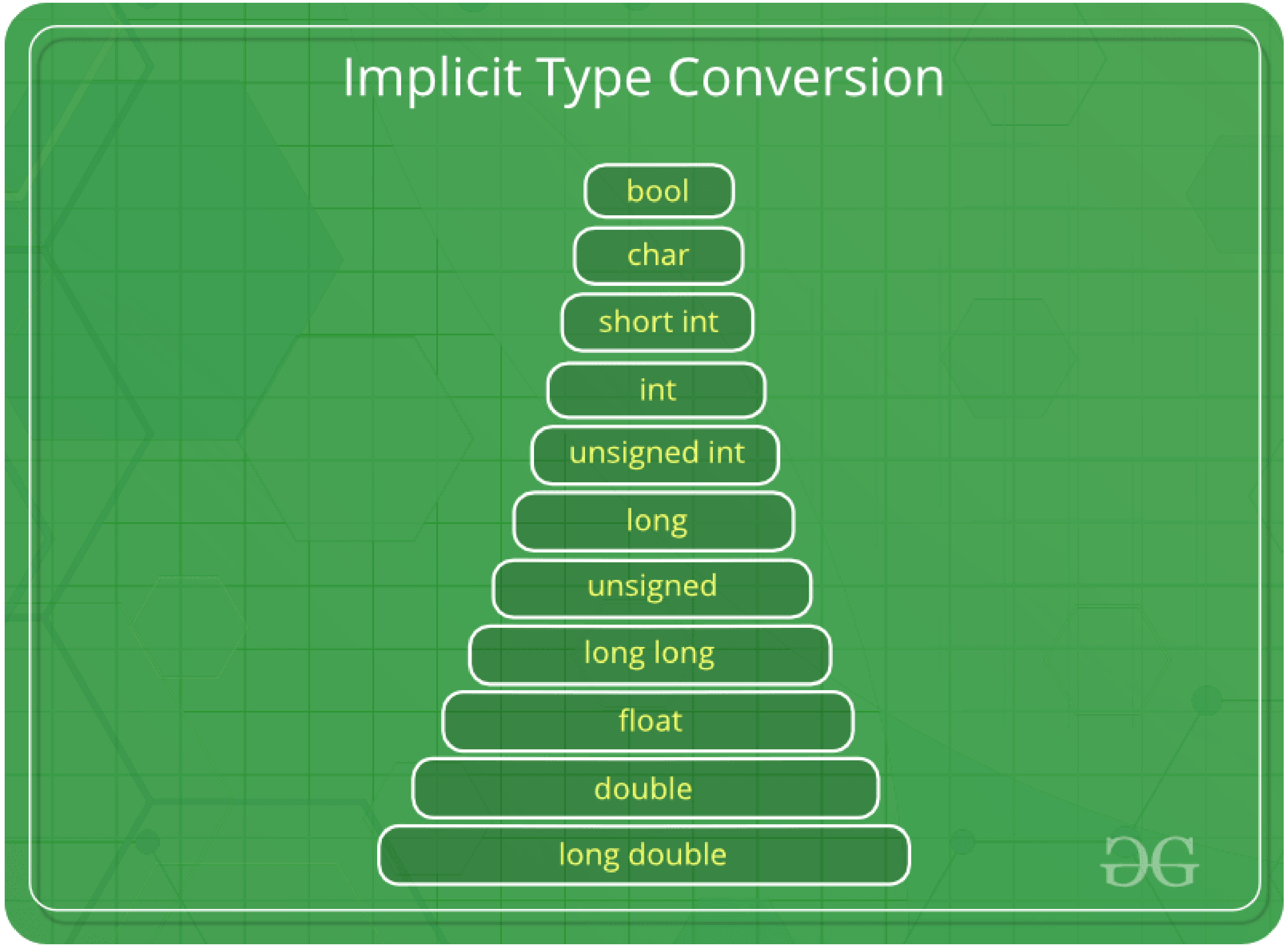


Type Conversion in C

- Type conversion in C is the process of converting one data type to another. The type conversion is only performed to those data types where conversion is possible. Type conversion is performed by a compiler. In type conversion, the destination data type can’t be smaller than the source data type. Type conversion is done at compile time.
- ***There are two types of Conversion:***
- **1. Implicit Type Conversion**
- **2. Explicit Type Conversion**

Implicit Type Conversion

Also known as ‘automatic type conversion’.
A. Done by the compiler on its own, without any external trigger from the user.
B. Generally takes place when in an expression more than one data type is present. In such conditions type conversion (type promotion) takes place to avoid loss of data.
C. All the data types of the variables are upgraded to the data type of the variable with the largest data type.
bool -> char -> short int -> int -> unsigned int -> long -> unsigned -> long long -> float -> double -> long double
D. It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long is implicitly converted to float).


Implicit Type Conversion
bool
char
short int
int
unsigned int
long
unsigned
long long
float
double
long double

```
// An example of implicit conversion
#include <stdio.h>
int main()
{
    int x = 10; // integer x
    char y = 'a'; // character c

    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);
    return 0;
}
```

Output

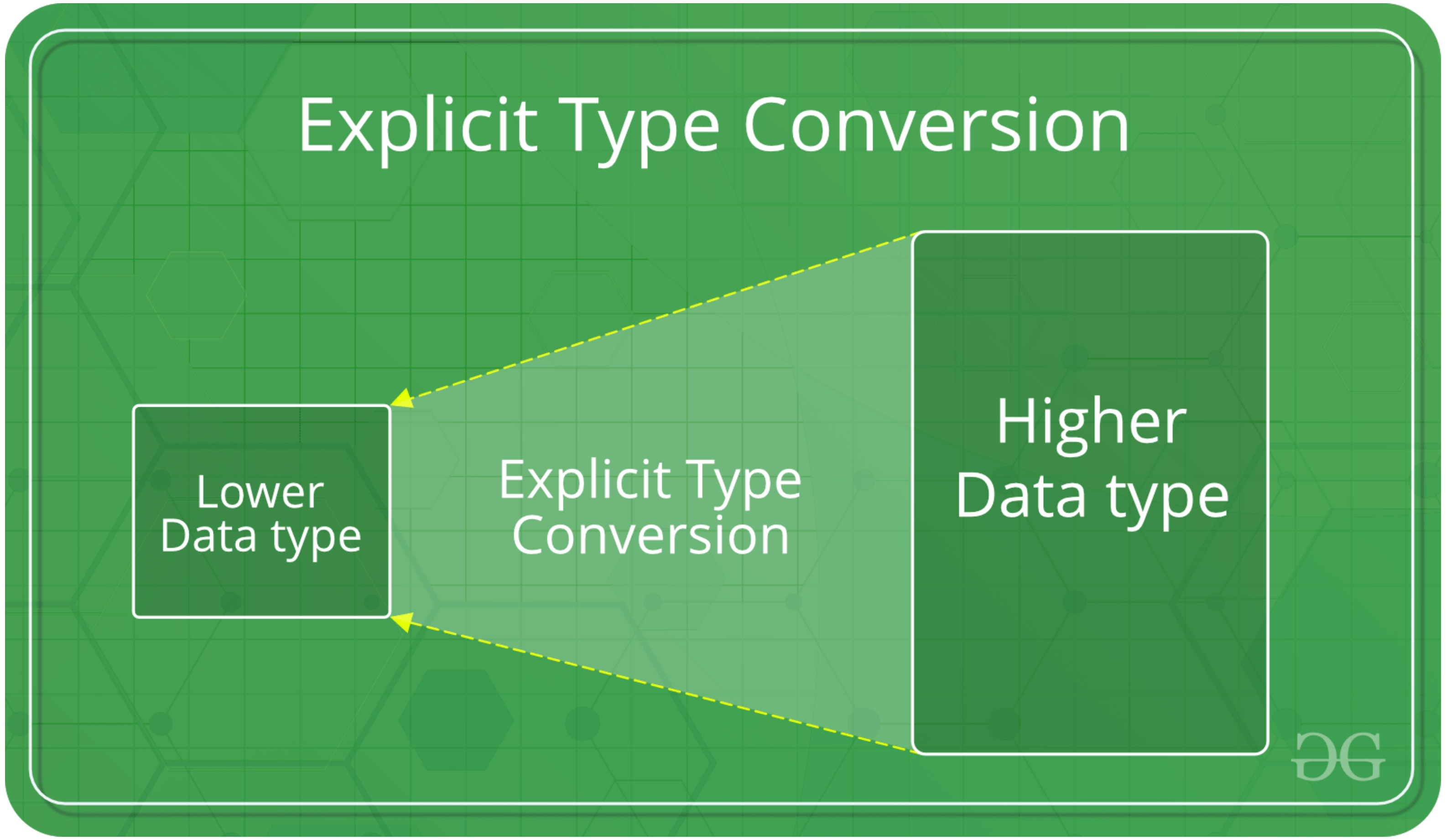
```
x = 107, z = 108.000000
```

Explicit Type Conversion

This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.

The syntax in C Programming:

(type) expressionType indicated the data type to which the final result is converted


Explicit Type Conversion
Lower Data type
Explicit Type Conversion
Higher Data type

```c
// C program to demonstrate explicit type casting
#include<stdio.h>

int main()
{
    double x = 1.2;

    // Explicit conversion from double to int
    int sum = (int)x + 1;

    printf("sum = %d", sum);

    return 0;
}
```

Output

```
sum = 2
```

What is modifier in C and different types of modifiers

- Modifiers are keywords in c which changes the meaning of basic data type in c. It specifies the amount of memory space to be allocated for a variable. Modifiers are prefixed with basic data types to modify the memory allocated for a variable. There are five data type modifiers in C Programming Language:

- long
- short
- signed
- unsigned
- long long

What is the difference between Character, Integer, Float and Double data types.

- **Character** : **Character data type** is used to store a character. A variable of character data type allocated only one byte of memory and can store only one character. Keyword char is used to declare variables of type character. Range of character(char) data type is -128 to 127.
  For Example: char ch = 'A';

- **Integer** : **Integer data type** is used to store a value of numeric type. Keyword int is used to declare variables of integer type. Memory size of a variable of integer data type is dependent on Operating System.For example size of an integer data type in a 32 bit computer is 4 bytes whereas size of integer data type in 16 bit computer is 2 bytes.
  For Example: int count = 10;

- **Float** : **Floating point data type** is used to store a value of decimal values. Memory size of a variable of floating point data type is dependent on Operating System. Keyword float is used to declare variables of floating data type. For example size of an floating point data type in a 16 bit computer is 4 bytes.
  For Example: float rate = 5.6;

- **Double** : **Double data type** is similar to floating data type except it provides up-to ten digit of precision and occupies eight bytes of memory.
  For Example: double d = 11676.2435676542;

Formatted I/O Functions

- Formatted I/O functions are used to take various inputs from the user and display multiple outputs to the user. These types of I/O functions can help to display the output to the user in different formats using the format specifiers. These I/O supports all data types like int, float, char, and many more.
- **Why they are called formatted I/O?**
- These functions are called formatted I/O functions because we can use format specifiers in these functions and hence, we can format these functions according to our needs.

List of some format specifiers-

S NO.	Format Specifier	Type	Description
1	%d	int/signed int	used for I/O signed integer value
2	%c	char	Used for I/O character value
3	%f	float	Used for I/O decimal floating-point value
4	%s	string	Used for I/O string/group of characters
5	%ld	long int	Used for I/O long signed integer value
6	%u	unsigned int	Used for I/O unsigned integer value
7	%i	unsigned int	used for the I/O integer value
8	%lf	double	Used for I/O fractional or floating data
9	%n	prints	prints nothing

The following formatted I/O functions will be discussed in this section-

- printf()
- scanf()
- sprintf()
- sscanf()
- **printf():**
- printf() function is used in a C program to display any value like float, integer, character, string, etc on the console screen. It is a pre-defined function that is already declared in the stdio.h(header file).
- **Syntax 1:**
- To display any variable value.
- ***printf("Format Specifier", var1, var2, …., varn);***

scanf():

•scanf() function is used in the C program for reading or taking any value from the keyboard by the user, these values can be of any data type like integer, float, character, string, and many more. This function is declared in stdio.h(header file), that’s why it is also a pre-defined function. In scanf() function we use &(address-of operator) which is used to store the variable value on the memory location of that variable.

•**Syntax:**

•***scanf(“Format Specifier”, &var1, &var2, …., &varn);***

sprintf():

•sprintf stands for **“string print”**. This function is similar to printf() function but this function prints the string into a character array instead of printing it on the console screen.

•**Syntax:**

•***sprintf(array_name, “format specifier”, variable_name);***

- **sscanf():**
- sscanf stands for **“string scanf”.** This function is similar to scanf() function but this function reads data from the string or character array instead of the console screen.
- **Syntax:**
- ***sscanf(array_name, “format specifier”, &variable_name);***

Unformatted Input/Output functions

- Unformatted I/O functions are used only for character data type or character array/string and cannot be used for any other datatype. These functions are used to read single input from the user at the console and it allows to display the value at the console.

Why they are called unformatted I/O?

- These functions are called unformatted I/O functions because we cannot use format specifiers in these functions and hence, cannot format these functions according to our needs.

The following unformatted I/O functions will be discussed in this section-

- getch()
- getche()
- getchar()
- putchar()
- gets()
- puts()
- putch()

getch():

getch() function reads a single character from the keyboard by the user but doesn’t display that character on the console screen and immediately returned without pressing enter key. This function is declared in conio.h(header file). getch() is also used for hold the screen.

•**Syntax:**

•***getch(); or***

•***variable-name = getch();***

getche():

getche() function reads a single character from the keyboard by the user and displays it on the console screen and immediately returns without pressing the enter key. This function is declared in conio.h(header file).

•**Syntax:**

•***getche();***

•***or***

•***variable_name = getche();***

getchar():

•The getchar() function is used to read only a first single character from the keyboard whether multiple characters is typed by the user and this function reads one character at one time until and unless the enter key is pressed. This function is declared in stdio.h(header file)

•**Syntax:**

•***Variable-name = getchar();***

putchar():

•The putchar() function is used to display a single character at a time by passing that character directly to it or by passing a variable that has already stored a character. This function is declared in stdio.h(header file)

•**Syntax:**

•***putchar(variable_name);***

gets():

•gets() function reads a group of characters or strings from the keyboard by the user and these characters get stored in a character array. This function allows us to write space-separated texts or strings. This function is declared in stdio.h(header file).

•**Syntax:**

•***char str[length of string in number]; //Declare a char type variable of any length***

•***gets(str);***

puts():

•In C programming puts() function is used to display a group of characters or strings which is already stored in a character array. This function is declared in stdio.h(header file).

•**Syntax:**

• ***puts(identifier_name);***

putch():

•putch() function is used to display a single character which is given by the user and that character prints at the current cursor location. This function is declared in conio.h(header file)

•**Syntax:**

•***putch(variable_name);***

Formatted I/O vs Unformatted I/O

S No.	Formatted I/O functions	Unformatted I/O functions
1	These functions allow us to take input or display output in the user’s desired format.	These functions do not allow to take input or display output in user desired format.
2	These functions support format specifiers.	These functions do not support format specifiers.
3	These are used for storing data more user friendly	These functions are not more user-friendly.
4	Here, we can use all data types.	Here, we can use only character and string data types.
5	printf(), scanf, sprintf() and sscanf() are examples of these functions.	getch(), getche(), gets() and puts(), are some examples of these functions.