Dr. SNS RAJALAKSHMI COLLEGE OF ARTS AND SCIENCE

(AUTONOMOUS)

Accredited by NAAC (Cycle- III) with 'A+' Grade

DEPARTMENT OF B.SC CS (GCD)

23UCU401 – PROGRAMMING IN C
UNIT- II

Subject : Programming in C
Department : GCD
Faculty : Mrs.K.Sangeetha

Unit 2 :

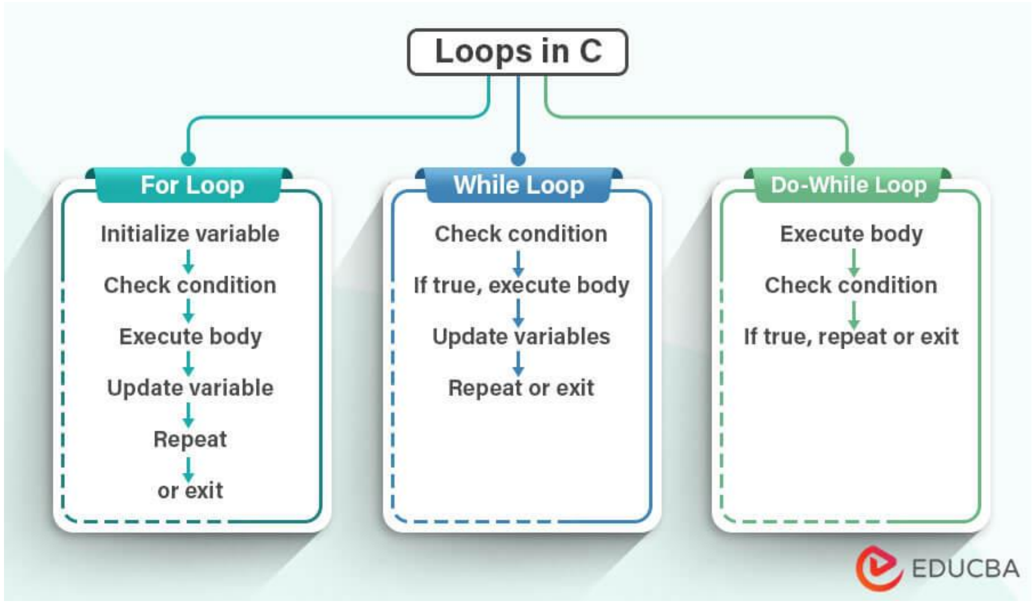
Loops - The while Loop - The for Loop - The Odd Loop - The break Statement - The continue Statement - The do-while Loop. The Case Control Structure: Decisions Using switch - switch Versus if-else Ladder - The goto Keyword.

1.Loops

Loops are a powerful tool in C programming that allow you to iterate over data and perform repetitive tasks.

There are three main types of loops in C:

- while loops
- for loops
- do-while loops.



- Loops in programming are used to execute a block of code repeatedly based on a condition.
- They help automate repetitive tasks and make programs more efficient.

2.The while Loop

- The while loop repeatedly executes a block of code as long as a specified condition is true.
- It's essential for tasks where the number of iterations is unknown in advance.

The syntax for a while loop is as follows:

```
while (condition) {
  // Code to be executed repeatedly
}
```

The `condition` is a Boolean expression. If the `condition` evaluates to `true`, the code inside the loop body will be executed. The loop will continue to execute as long as the `condition` evaluates to `true`.

For example, the following code uses a while loop to print the numbers from 1 to 10:

```
int main() {
  int i = 1;
  while (i <= 10) {
    printf("%d ", i);
    i++;
  }

  return 0;
}
```

Output:

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

3.The for Loop

The for loop is similar to the while loop, but it is more concise and easier to read. The syntax for a for loop is as follows:

```
for (initialization; condition; increment) {
  // Code to be executed repeatedly
}
```

The `initialization` expression is executed at the beginning of the loop. The `condition` expression is evaluated before each iteration of the loop. If the `condition` evaluates to `true`, the code inside the loop body will be executed. The `increment` expression is executed at the end of each iteration of the loop.

For example, the following code uses a for loop to print the numbers from 1 to 10:

```
int main() {
  for (int i = 1; i <= 10; i++) {
    printf("%d ", i);
  }

  return 0;
}
```

This code is equivalent to the while loop example previously, but it is more concise and easier to read.

4.The Odd Loop

The odd loop is a specialized type of for loop that is used to iterate over odd numbers. The syntax for an odd loop is as follows:

```
for (int i = start; i <= end; i += 2) {
  // Code to be executed repeatedly
}
```

The `start` and `end` expressions are the range of numbers to iterate over. The `i += 2` expression increments the loop counter by 2, ensuring that only odd numbers are iterated over.

For example, the following code uses an odd loop to print all of the odd numbers from 1 to 10:

```
int main() {
  for (int i = 1; i <= 10; i += 2) {
    printf("%d ", i);
  }

  return 0;
}
```

Output:

```
1 3 5 7 9
```

5.The break Statement

The break statement is used to exit a loop. The syntax for the break statement is as follows:

Syntax :

```
break;
```

- When the break statement is encountered, the loop will immediately exit.

- The break statement is used to exit a loop prematurely.
- It's typically used within loops or switch statements when a certain condition is met.

For example, the following code uses the break statement to exit the loop once the number 10 is reached:

```
int main() {
  for (int i = 1; i <= 10; i++) {
    if (i == 10) {
      break;
    }

    printf("%d ", i);
  }

  return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9
```

6.The continue Statement

The continue statement is used to skip the remaining code in the current iteration of a loop and continue to the next iteration.

The syntax for the continue statement is as follows:

```
continue;
```

- The continue statement is used to skip the rest of the current iteration of a loop.
- It's helpful when you want to continue with the next iteration without executing the remaining code in the current iteration.

Here is a simple C program to use the continue statement:

```
#include <stdio.h>

int main() {
  int i;

  for (i = 1; i <= 10; i++) {
    if (i % 2 == 0) {
      continue;
    }

    printf("%d ", i);
  }

  return 0;
}
```

This program uses a for loop to iterate over the numbers from 1 to 10. If the number is even, the continue statement is used to skip the rest of the loop body and continue to the next iteration. Otherwise, the number is printed to the console.

1 3 5 7 9ompile and run this program, it will print the following output to the console:

7.The do-while Loop

The do-while loop is a post-test loop in C, which means that the loop body is executed at least once,

The syntax for a do-while loop is as follows:

```
do {
  // Loop body
} while (condition);
```

The `condition` is a Boolean expression. The loop body will be executed at least once, and then the `condition` will be evaluated. If the `condition` evaluates to true, the loop body will be executed again. The loop will continue to execute as long as the `condition` evaluates to true.

Here is an example of a do-while loop:

```
int main() {
  int number = 1;

  do {
    printf("%d ", number);
    number++;
  } while (number <= 10);

  return 0;
}
```

This program uses a do-while loop to print the numbers from 1 to 10 to the console. The loop body will be executed at least once, even if the initial value of `number` is greater than 10.

When you compile and run this program, it will print the following output to the console:

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

8.The Case Control Structure

The case control structure in C is a decision-making structure that allows you to execute different blocks of code based on the value of a variable.

The case control structure is similar to the if-else statement, but it is more powerful and flexible.

Here is an example of a case control structure:

```
int main() {
  int dayOfWeek = 1;

  switch (dayOfWeek) {
    case 1:
      printf("Monday\n");
      break;
    case 2:
      printf("Tuesday\n");
      break;
    case 3:
      printf("Wednesday\n");
      break;
    case 4:
      printf("Thursday\n");
      break;
    default:
      printf("Invalid day of the week.\n");
  }
  return 0;
}
```

9.Decisions Using switch

The switch statement in C is a decision-making structure that allows you to execute different blocks of code based on the value of a variable. It is similar to the if-else statement, but it is more concise and efficient for making decisions with multiple cases.

```
switch (variable) {
  case value1:
    // Code to be executed if variable == value1
    break;
  case value2:
    // Code to be executed if variable == value2
    break;
  ...
  default:
    // Code to be executed if variable does not match any of the other cases
}
```

e `variable` is the variable whose value is being compared. The `value1`, `value2`, and so on, are the values that the variable is being compared to. The `break` statement is used to exit the switch statement after a case has been matched. The `default` case is executed if the variable does not match any of the other cases.

example:

```
int main() {
  int examScore = 85;

  switch (examScore) {
    case 90 ... 100:
      printf("A\n");
      break;
    case 80 ... 89:
      printf("B\n");
      break;
    case 70 ... 79:
      printf("C\n");
      break;
    case 60 ... 69:
      printf("D\n");
      break;
    default:
      printf("F\n");
  }
  return 0;
}
```

10.switch Versus if-else Ladder

The switch statement and the if-else ladder are both decision-making structures in C. They can both be used to execute different blocks of code based on the value of a variable. However, there are some key differences between the two:

Switch Statement

- The switch statement is more concise and efficient for making decisions with multiple cases.
- The switch statement can only compare the value of the variable to integer or character constants.
- The switch statement must have a default case.

If-Else Ladder

- The if-else ladder is more flexible than the switch statement. It can be used to compare the value of the variable to any type of data, including floating-point numbers and strings.
- The if-else ladder does not need to have a default case.

Which One to Use?

In general, you should use the switch statement for making decisions with multiple cases. It is more concise and efficient than the if-else ladder. However, if you need to compare the value of the variable to a floating-point number or a string, or if you do not need a default case, then you should use the if-else ladder.

Here is an example of an if-else ladder:

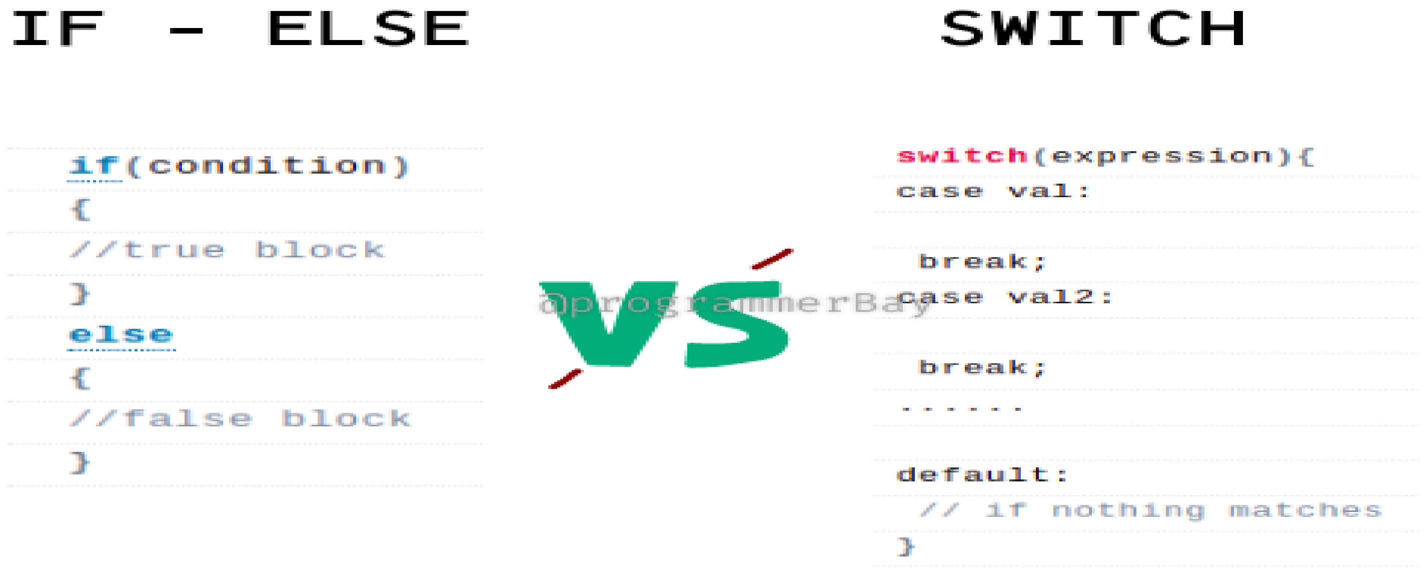
```
int main() {
  int examScore = 85;

  if (examScore >= 90) {
    printf("A\n");
  } else if (examScore >= 80) {
    printf("B\n");
  } else if (examScore >= 70) {
    printf("C\n");
  } else if (examScore >= 60) {
    printf("D\n");
  } else {
    printf("F\n");
  }

  return 0;
}
```

Which one to use depends on the specific needs of your program. If you are making a decision with multiple cases, then the switch statement is more concise and efficient.

If you need to compare the value of the variable to a floating-point number or a string, or if you do not need a default case, then you should use the if-else ladder.



```
if(condition)
{
//true block
}
else
{
//false block
}
```

```
switch(expression){
case val:

 break;
case val2:

 break;
......

default:
 // if nothing matches
}
```

11.The goto Keyword.

The goto keyword in C is a jump statement that allows you to transfer program control to a labeled statement. It is a powerful tool, but it can also be dangerous if used incorrectly.

The syntax for the goto statement is as follows:

```
goto label;
```

The `label` is the identifier of the statement to which you want to transfer control. The `label` can be placed anywhere in the program, above or below the goto statement.

Here is an example of how to use the goto statement:

```
int main() {
  int i;

  for (i = 0; i < 10; i++) {
    if (i == 5) {
      goto my_label;
    }

    printf("%d ", i);
  }

  my_label:
    printf("This is my label.\n");

  return 0;
}
```

When you compile and run this program, it will print the following output to the console:

```
0 1 2 3 4 This is my label.
```