week 2_2:

loops

rajin teaches programming

– OVERVIEW:

Looping structures are **fundamental** in programming, allowing the **repetition** of a **block** of code based on certain **conditions.** They provide **efficiency** and **flexibility** in handling **repetitive** tasks. In this handout, we'll delve into the three types of loops, namely the concepts of **for** loops, **while** loops, and **do-while** loops, exploring their syntax, purpose, and best practices.

FOR LOOPS:

– WHAT ARE FOR LOOPS?

A **for loop** is used to execute a block of code **repeatedly** for a **fixed** number of times. It is ideal when the **number** of **iterations** is **known** in **advance.**

– SYNTAX:

FOR-LOOP

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

In the block of code above, the **initialization** part takes an **iterator**, which is a new **integer** variable.
Then, the **condition** is checked to **verify.** If it **holds**, the **block** of code is then **executed.** Afterwards, the value of the iterator is **incremented** by **1** (in most cases).
Then, the **condition** is checked **again.** If it **holds**, the **block** of code is **executed.**
This **continues until** the condition **does not** hold **true**, at which point it **exits** the loop.

rajin teaches programming

```c
#include<stdio.h>

int main() {

    printf("All the numbers from 1 to 10 are: ");

    for (int i=1; i<=10; i++) {

        printf("%d ", i);
    }

    printf("\n");

    return 0;
}
```

an example of a for loop that prints all the integers from 1 to 10.

WHILE LOOPS:

– WHAT ARE WHILE LOOPS?

A **while loop** executes a block of code **repeatedly as long as** a **specified condition** is **true.** It is **suitable** when the number of **iterations** is **not predetermined.**

WHILE-LOOP

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

In the block of code above, the **condition** is **checked,** and the block of code **inside** the section is **executed as long as** the **condition** is **true.** As you can imagine from experience with the for loop above, the condition must be with an **iterating variable,** and the **iterator** must be **incremented in** the **code block** itself. Furthermore, the **iterating variable must** be **declared before** the while loop begins.

rajin teaches programming

```c
#include<stdio.h>

int main() {

    int i = 0;

    while (i<10) {

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

    return 0;
}
```

iterator declared in advance

condition

iterator incremented

a similar example of a while loop that prints all the integers from 0 to 9

DO-WHILE LOOPS:

- WHAT ARE DO-WHILE LOOPS?

A **do-while loop** is **similar** to a **while loop** but **guarantees** the **execution** of the block of code **at least once, even if** the **condition** is **initially false.**

DO-WHILE-LOOP

```c
do {
    // Code to be executed
} while (condition);
```

In the block of code above, the code is **initially executed,** and the **condition** is **checked afterwards.** As you may expect, the **constraints** of **while loops apply** here **too,** as the **iterator** must be **declared** in **advance,** and **incremented within** the **code block.** They key **differentiator** here is that the **code** is **executed at least once regardless** of the **condition being true.** This may be useful in programs in which **input** needs to be taken at least **once.**

rajin teaches programming

```c
#include<stdio.h>

int main() {

    int i = 0;

    do {

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

    } while(i<10);

    return 0;
}
```

a similar example of a do-while loop that prints all the integers from 0 to 9

FURTHER CONCEPTS:

– INFINITE LOOPS:

An **infinite loop** occurs when the loop **condition always** evaluates to **true,** causing the loop to run **indefinitely.** Although this may not seem too useful, it is actually an **easier alternative** when we **do not know** the **number of inputs** that need to be taken.

```c
#include<stdio.h>

int main() {

    int input;

    while(1) {

        scanf("%d", &input);

        if (input<0) {

            break;
        }
    }

    return 0;
}
```

a similar example of an infinite while loop that takes input until a negative number is entered

rajin teaches programming

– BREAK STATEMENTS:

Although discussed and used briefly in conditional statements, **break statements** warrant their own section when learning about loops for their importance. As you may already know, break statements help you exit a code-block when used in conditional statements. However, for **loops**, they **function** a little **differently.** The **break statement** is **used** to **exit** a loop **prematurely based** on **certain conditions, providing control** over **loop execution.** In the infinite loop example, you can see that it does **not just exit** the **conditional statement** code **block, but breaks out** of the **loop entirely.**

```
Developer -
#include<stdio.h>

int main() {

    for (int i=1; i<=10; i++){

        if(i == 5) {

            break;
        }

        printf("%d ", i);
    }

    return 0;
}
```

an example of a break statement being used to exit a loop prematurely. can you guess the output?

SUMMARY:

Loops **iterate** over a **block** of **code multiple times,** based on **specified conditions.** They **streamline repetitive tasks** and **enhance** code **efficiency. FOR** loops are **ideal** for a **fixed** number of iterations. **WHILE** loops are suitable for situations where the number of **iterations** is **not predetermined. DO-WHILE** loops **ensure** the **execution** of the loop body **at least once.**

rajin teaches programming

next class 2_3:

patterns

rajin teaches programming

reach out for classes at: rajin.khan2001@gmail.com