1. What is the function of control statements? What form can a control expression be?
   soln:
   Main Functions of Control Statements:

   - **Decision-Making** – Control which block of code runs based on conditions.
     - Example: if-else, switch.

   - **Looping** – Repeat code execution until a condition is met.
     - Example: for, while, do-while.

   - **Branching and Jumping** – Alter the normal flow using jumps.
     - Example: break, continue, goto, return.

Control Statement	Control Expression Type	Example
if, else if	Boolean condition (true/false, 0 or nonzero)	**if (x > 10)**
switch	Integer or character constant/expression	**switch (choice)**
for loop	Initialization, condition, update	**for (int i = 0; i < 10; ++i)**
while loop	Boolean condition	**while(x < 5)**
do-while loop	Boolean condition (evaluated after execution)	**do { ... } while (x < 5);**
break, continue	No control expression (acts based on conditions)	**if (x == 5) break;**
goto	Label-based jump	**goto label;**

2. How many kinds of selection statements are there? What are their application scenarios?
   soln:
   There are **two types** of selection statements:

- if statement (Simple, Nested, and if-else-if)
- switch statement

Selection Type	Application Scenario
if	Checking if a number is even **(if (num % 2 == 0)).**

if-else	Checking login credentials
	if (password == correctPassword)){ // code }
	else { // code }

if-else-if ladder	Assigning letter grades based on marks **(if (marks >= 90) ... else if (marks >= 80)).**
Nested **if**	Checking if a number is positive and even **(if (num > 0) { if (num % 2 == 0) }).**
switch	Menu-driven programs **switch(choice) { case 1: Start; case 2: Exit; }**

3. What should we pay attention to in the switch statement?
   soln:

Aspect	Common Issues	Best Practices

Forgetting break	Omitting break causes fall-through, leading to unintended execution of the next case.	Always use break unless fall-through is intentional (and document it).
Unintentional Fall-Through	Without break, execution continues into the next case.	If intentional, use [[fallthrough]]; (C17) or a comment (// fall-through).
Not Handling the default Case	Missing default can lead to unexpected behavior if none of the case conditions match.	Always include a default case to handle unexpected values.
Using Duplicate case Values	Defining two case labels with the same value causes a compilation error.	Ensure all case labels have unique values.
Using switch for Floating-Point Values	switch only works with integers and characters, not float or double	Use if-else for floating-point comparisons.
Scope of Variables in case Blocks	Declaring variables inside case without { } leads to errors.	Use { } to define local scope inside case.

4. What is dangling-else? Illustration

   soln: The dangling else problem occurs when there are multiple if statements without explicit { } braces, and the else clause ambiguously associates with **the nearest unmatched if.**

```c
#include <stdio.h>

int main() {
    int x = 5, y = 10;
```

```
    if (x < 3)
        if (y > 5)
            printf("Both conditions are true\n");
    else
        printf("Else block\n");
    // Anyone might think the else belongs to the first if
(x > 3).
    // But in reality, it gets associated with the closest
unmatched if, which is if (y > 5)


    return 0;
}
```

this code output is : (blank)

5. Pointers-On-C.pdf 4.13 question 4
slon:

```
#include<stdio.h>

int main(){
    int x;

    // using ternary operator
    x > 10 ? : printf("This is the else part\n");


    if (x > 10);  // Empty statement no action
    else {
        printf("This is the else part\n");
    }

    //using goto
    if (x > 10)
```

```
    goto skip_else;

    printf("This is the else part\n");

    skip_else:
    // Continue execution here if condition is true

}
```

6. How many kinds of loop statements are there? What are their application scenarios?
   soln: In C, there are three types of loops:

   - **for loop** – Best for loops with a known number of iterations.
   - **while loop** – Best for loops with an unknown number of iterations but a clear condition.
   - **do-while loop** – Best when the loop body must execute at least once before checking the condition.

Loop Type	Syntax	Best Used When	Example Scenario
for loop	for(initialization; condition; update) { ... }	Iterations are **fixed** or known in advance.	Iterating over an **array**, running a loop **N times**.
while loop	while(condition) { ... }	Iterations are **unknown** but depend on a condition.	Reading input until a specific condition is met, like waiting for user input (while(intput != -1)).

do-while loop	do { ... } while(condition);	The loop **must execute at least once**, regardless of condition.	Menu-driven programs where the menu should display at least once before checking the exit condition.

7. What is the difference between break, continue and return when we use them in a nested loop? Illustration
soln:

Statement	Effect	Scope in Nested Loops	Use Case
break	Exits the **current loop**	Affects **only the innermost loop**	When need to **exit a loop early**
continue	Skips the **current iteration**	Affects **only the loop where it appears**	When need to **skip some iterations**
return	Exits the **entire function**	Stops all loops and function execution	When need to **end execution immediately**

```c
#include <stdio.h>

void function(){

    for(int i = 0; i < 10; ++i){
        for(int j = 0; j < 10; ++j){
            if(i+j <= 0){
                return; // return from here and function
and loop execution stop
            }
        }
```

```
    }
    printf("Left part of this code\n");
}

int main() {

    // break
    for (int i = 1; i <= 3; i++) {
        for (int j = 1; j <= 3; j++) {
            if (j == 2) break;  // Exits inner loop when j
== 2
            printf("i = %d, j = %d\n", i, j);
        }
    }

    // continue
    for (int i = 1; i <= 3; i++) {
        for (int j = 1; j <= 3; j++) {
            if (j == 2) continue;;  // Skip left of the
code of inner loop when j == 2
            printf("i = %d, j = %d\n", i, j);
        }
    }
    function();
    return 0;
}
```

8. Pointers-On-C.pdf 5.8 question 8
   soln:

```
a = f1(x);
b = f2(x + a);

while ((c = f3(a, b)) > 0) {
   statements;
   a = f1(++x);
   b = f2(x + a);
}
```

9. What are the advantages and disadvantages of the goto statement?
   soln:
   - advantages:
     - **Simplifies Error Handling (Early Exits):** In deeply nested loops or functions, goto can help exit from multiple levels at once.
     - **Can Improve Performance in Low-Level Code :** In performance-critical code, especially embedded systems or OS kernels, goto can optimize control flow by reducing unnecessary function calls.
     - **Useful in Finite State Machines:** Some finite state machines and parsers rely on goto to switch between states efficiently.

   - disadvantages:
     - **Makes Code Hard to Read :** Overuse of goto results in unstructured code that is difficult to follow.
     - **Hard to Debug and Maintain :** Debugging goto-heavy code is difficult since it jumps across the program unpredictably.
     - **Can Be Replaced with Structured Control Flow :** Loops (for, while) and conditionals (if, switch) make code more structured.