Programming in C

Chapter: Decision Control and Looping Statements

Study Notes for Engineering / Polytechnic Students

2.1 Decision Making and Branching Statements

In C programming, a program does not always execute every statement in the order it is written. Many real-world problems require a decision to be taken based on a condition, and the program must choose one path of execution over another. This ability of a program to alter its flow depending on a condition is called decision making or branching. C provides several constructs for this purpose, the most important being the if statement and the switch statement.

The if Statement

The if statement is the most basic decision-making statement in C. It evaluates a condition (an expression that results in true or non-zero, or false or zero) and executes a block of code only when the condition is true. If the condition is false, the block is simply skipped and control moves to the statement following the if block.

```
if (condition)
{
    // statements executed only if condition is true
}
```

if-else Statement

The if-else statement extends the plain if statement by providing an alternative block of code that executes when the condition is false. This ensures that exactly one of the two blocks always executes, giving the program a two-way branch.

```
if (condition)
{
    // executes when condition is true
}
else
{
    // executes when condition is false
}
```

else-if Ladder

When a program needs to test a series of conditions and execute a different block for each case, the else-if ladder is used. Each condition is tested in order from top to bottom, and as soon as one condition evaluates to true, its corresponding block executes and the rest of the ladder is skipped. If none of the conditions are true, the final else block (if present) executes as the default case.

```
if (condition1)
{
    // block 1
}
else if (condition2)
```

```
{
    // block 2
}
else if (condition3)
{
    // block 3
}
else
{
    // default block
}
```

A common application of the else-if ladder is grading systems, where marks are compared against several ranges to assign a grade such as A, B, C, or F.

Nested if-else

When an if or if-else statement is placed inside the body of another if or else block, it is called nested if-else. Nesting is useful when a decision depends on more than one condition that must be checked in stages, rather than all at once. Care must be taken with indentation and matching braces, since deeply nested conditions can quickly become difficult to read and debug.

```
if (condition1)
{
    if (condition2)
    {
        // executes when both condition1 and condition2 are true
    }
    else
    {
        // executes when condition1 is true but condition2 is false
    }
}
```

Switch Case Statement

The switch statement is a multi-way branching statement used as a cleaner alternative to a long else-if ladder when a single variable or expression is compared against a fixed set of constant values. The expression is evaluated once, and control jumps directly to the matching case label. Each case block should normally end with a break statement to prevent execution from falling through into the next case. An optional default case handles any value that does not match the listed cases.

```
switch (expression)
{
    case value1:
        // statements
        break;
    case value2:
        // statements
        break;
    default:
        // statements executed when no case matches
}
```

Key points about switch statements:

- The switch expression must evaluate to an integer or character type; it cannot be a floating-point value or a string.
- Case labels must be constant expressions and must all be distinct.
- Omitting break causes fall-through, where execution continues into the next case regardless of its label — this is sometimes used deliberately to group cases together.
- The default case is optional and can be placed anywhere in the switch block, though convention places it last.
- switch is generally faster and more readable than a long else-if ladder when many discrete values must be checked.

2.2 Iterative / Loop Statements

Looping statements allow a block of code to be executed repeatedly as long as a specified condition holds true. Loops are essential for tasks such as processing arrays, repeating input operations, and performing calculations over a range of values, without having to rewrite the same statements multiple times.

Entry Controlled and Exit Controlled Loops

Loops in C are classified into two categories based on when the controlling condition is tested:

- Entry controlled loop: The condition is tested before the loop body executes. If the condition is false at the very first check, the body may not execute even once. The for loop and the while loop are entry controlled loops.
- Exit controlled loop: The condition is tested after the loop body executes. This guarantees that the body executes at least once, regardless of whether the condition is true or false. The do-while loop is the only exit controlled loop in C.

Difference between entry controlled and exit controlled loops:

- Entry controlled loops check the condition first, so the loop body may execute zero times; exit controlled loops check the condition last, so the body always executes at least once.
- while and for loops are entry controlled; do-while is exit controlled.
- In do-while, the syntax places the condition after the body and ends with a semicolon, unlike while and for.

while Loop

The while loop is an entry controlled loop that repeats a block of statements as long as the given condition remains true. The condition is checked before each iteration, including the first one.

```
while (condition)
{
    // statements
    // update expression to eventually make condition false
}
```

do-while Loop

The do-while loop is an exit controlled loop. The body of the loop is executed first, and the condition is evaluated afterward. If the condition is true, the loop repeats; otherwise it terminates. Because the check happens at the end, the loop body is guaranteed to run at least once.

```
do
{
    // statements
} while (condition);
```

for Loop

The for loop is an entry controlled loop that combines initialization, condition testing, and updating in a single line, making it the most compact and commonly used loop for situations where the number of iterations is known in advance.

```
for (initialization; condition; update)
{
    // statements
}
```

Here, the initialization expression executes once at the start, the condition is checked before every iteration, and the update expression executes at the end of every iteration, after which the condition is checked again.

Break and Continue Statements

- break statement: Immediately terminates the innermost loop (or switch statement) in which it appears, and transfers control to the statement immediately following that loop or switch.
- continue statement: Skips the remaining statements in the current iteration of the loop and moves control directly to the condition check (in while and do-while) or to the update expression (in for), thereby beginning the next iteration.

Both break and continue are jump statements used to alter the normal flow of a loop, but while break exits the loop entirely, continue only skips the current pass and allows the loop to keep running.

Conditional and Unconditional goto Statement

The goto statement transfers control to a labelled statement elsewhere in the same function, bypassing the normal sequential flow. A label is an identifier followed by a colon, placed before the target statement.

```
goto label;
...
label:
    statement;
```

- Unconditional goto: The jump always takes place whenever the goto statement is executed, with no condition attached, e.g. a plain goto label; on its own line.
- Conditional goto: The goto statement is placed inside an if statement, so the jump occurs only when the associated condition is true, e.g. if (condition) goto label;

The use of goto is generally discouraged in modern structured programming because it can make code difficult to read, follow, and debug, especially when overused. Loops, break, and continue can achieve the same effect in a more structured and readable manner, so goto is typically reserved for rare cases such as breaking out of deeply nested loops in one step.

Nested Loop Structure

A nested loop is a loop placed inside the body of another loop. The loop that contains another loop is called the outer loop, and the loop inside it is called the inner loop. For every single iteration of the outer loop, the inner loop runs through all of its own iterations completely. Nested loops are widely used for working with two-dimensional data, such as matrices, and for generating patterns.

```
for (i = 1; i <= rows; i++)
{
    for (j = 1; j <= cols; j++)
    {
        // inner loop body runs 'cols' times
        // for every single value of i
    }
}
```

3.1 Arrays

An array is a collection of elements of the same data type, stored in contiguous memory locations, and referred to by a single common name. Each element of an array is accessed using an index, also called a subscript. Because an array behaves like a single variable holding multiple values, it is often called a subscripted variable.

Advantages of Subscripted Variables / Arrays

- A single array name can represent many related values, avoiding the need to declare a separate variable for each value.
- Elements are stored in contiguous memory, allowing fast, direct, and random access to any element using its index.
- Arrays make it possible to process large sets of data efficiently using loops, instead of writing repetitive code for each value.
- They simplify sorting, searching, and other common data-processing operations by giving structured access to a sequence of values.
- Arrays form the basis for more advanced data structures, such as matrices, strings, stacks, and queues.

One Dimensional Arrays

A one dimensional array stores a linear list of elements. It is declared by specifying the data type, the array name, and the number of elements (the size) in square brackets.

```
data_type array_name[size];

// Declaration
int marks[5];

// Declaration with initialization
int marks[5] = {70, 65, 90, 55, 82};

// Size can be omitted when initializing directly
int marks[] = {70, 65, 90, 55, 82};
```

Two Dimensional Arrays

A two dimensional array stores data in a table-like structure of rows and columns, and is commonly used to represent matrices. It requires two subscripts — one for the row and one for the column.

```
data_type array_name[rows][columns];

// Declaration
int matrix[3][3];

// Declaration with initialization
int matrix[2][3] = { {1, 2, 3}, {4, 5, 6} };
```

Character Arrays

A character array stores a sequence of characters and is the fundamental way strings are represented in C. When initialized with a string constant, the compiler automatically appends a null character ('\0') at the end to mark where the string terminates.

```
char name[20];

// Initialization with a string literal
char name[20] = "Engineering";

// Initialization as a list of characters (null terminator added manually)
char name[] = {'C', ' ', 'L', 'a', 'n', 'g', '\0'};
```

Accessing Array Elements

In C, array indexing always begins at zero. This means that for an array of size n, valid indices run from 0 to n minus 1. An individual element is accessed by writing the array name followed by the required index in square brackets.

```
int marks[5] = {70, 65, 90, 55, 82};

printf("%d", marks[0];   // prints 70, the first element
printf("%d", marks[4];   // prints 82, the last element
marks[2] = 100;          // modifies the third element

// Accessing a two dimensional array element
printf("%d", matrix[1][2];   // row index 1, column index 2
```

Loops, especially the for loop, are typically used to access or process every element of an array systematically, since the loop counter can double as the array index.

3.2 Strings and String Handling

In C, a string is simply an array of characters terminated by a null character ('\0'). There is no separate built-in string data type; instead, strings are handled through character arrays and, together with pointers, through the standard string-handling library declared in the header file string.h.

Declaration and Initialization of String Variables

```
// Declaration only
char city[30];

// Declaration with initialization using a string literal
```

```
char city[30] = "Kolkata";

// Reading a string from the keyboard
scanf("%s", city);       // no & needed, since the array name is already an address

// Reading a string that may contain spaces
fgets(city, sizeof(city), stdin);
```

The compiler reserves one extra byte beyond the visible characters of a string literal to store the terminating null character, so an array meant to hold a string of length n must be declared with a size of at least n plus 1.

String Handling Functions from the Standard Library

The header file string.h provides several ready-made functions for common string operations, removing the need to write these routines manually every time.

- strlen(str): Returns the length of the string, that is, the number of characters before the terminating null character. It does not count the null character itself.
- strcpy(dest, src): Copies the string src, including its terminating null character, into the character array dest. The destination array must be large enough to hold the copied string.
- strcat(dest, src): Appends (concatenates) the string src to the end of the string dest, and places a single null character at the new end. The dest array must have enough spare space to accommodate the added characters.
- strcmp(str1, str2): Compares two strings character by character and returns zero if they are identical, a negative value if str1 is alphabetically less than str2, and a positive value if str1 is alphabetically greater than str2.

```
#include <string.h>

char a[20] = "Hello";
char b[20] = "World";

printf("%d", strlen(a));      // 5
strcpy(b, a);                 // b now holds "Hello"
strcat(a, b);                 // a now holds "HelloHello"
printf("%d", strcmp(a, b));   // non-zero, since a and b now differ
```

Extracting a Substring — Left, Right, and Middle

The standard library does not provide direct left(), right(), or mid() functions as found in some other languages; the same effect is achieved in C using a combination of strncpy(), array indexing, and pointer arithmetic, along with strlen() to determine the total length of the source string.

- Left substring: Copy the required number of characters starting from index 0 of the source string using strncpy(), and then manually place a null character at the end of the destination array.
- Right substring: Calculate the starting index as (total length minus the number of characters required), and copy from that position to the end of the string.
- Middle substring: Copy the required number of characters starting from a chosen offset within the string, again followed by a manually placed null terminator.

```
char str[] = "Programming";
char result[20];
```

```
int len = strlen(str);         // 11

// Left 4 characters -> "Prog"
strncpy(result, str, 4);
result[4] = '\0';

// Right 4 characters -> "ming"
strncpy(result, str + (len - 4), 4);
result[4] = '\0';

// Middle 4 characters starting at offset 3 -> "gram"
strncpy(result, str + 3, 4);
result[4] = '\0';
```

Replacement of String Characters

Since C strings are simply character arrays, an individual character within a string can be replaced directly through array indexing, without needing any library function. Replacing an existing character does not change the length of the string, since one character is simply substituted for another at the same position.

```
char str[] = "Programming";
str[0] = 'p';        // "programming"

// Replacing every occurrence of one character with another
for (int i = 0; str[i] != '\0'; i++)
{
    if (str[i] == 'm')
        str[i] = 'M';
}
// result: "prograMMing"
```

Concatenation of Two Strings

Concatenation means joining one string to the end of another to form a single, combined string. In C, this is most commonly done using the strcat() function from string.h, which appends its second argument to the first.

```
char first[30] = "Web";
char second[] = "ByArun";

strcat(first, second);    // first now holds "WebByArun"
printf("%s", first);
```

Care must always be taken to ensure that the destination array passed to strcat() is large enough to hold both original strings together, along with the terminating null character; otherwise the operation may write beyond the bounds of the array and corrupt adjacent memory.

Multiple Choice Questions (15)

1. Which statement is used to test a condition and execute one of two blocks of code?

a) for
b) if-else
c) while
d) goto

2. In an else-if ladder, what happens once a condition evaluates to true?

a) All remaining conditions are still checked
b) The program terminates
c) Its block executes and the rest of the ladder is skipped
d) The loop restarts

3. Which of the following data types CANNOT be used directly as a switch expression?

a) int
b) char
c) float
d) enum

4. What is the purpose of the break statement inside a switch case?

a) It starts the next case
b) It prevents fall-through into the next case
c) It exits the entire program
d) It repeats the current case

5. Which loop is an exit controlled loop?

a) for
b) while
c) do-while
d) nested for

6. In a for loop for (i = 1; i <= 5; i++), which part executes first?

a) condition
b) update expression
c) initialization
d) loop body

7. What does the continue statement do inside a loop?

a) Terminates the loop immediately
b) Skips the rest of the current iteration and moves to the next one
c) Restarts the entire program
d) Exits the switch statement

8. A goto statement placed inside an if block is an example of:

a) Unconditional goto

b) Conditional goto

c) Nested goto

d) Switch statement

9. In a nested loop, for every single iteration of the outer loop, the inner loop:

a) Executes exactly once

b) Does not execute at all

c) Completes all of its own iterations

d) Executes only if a condition is true

10. In C, array indices always start from:

a) 1

b) -1

c) 0

d) The array size

11. Which of the following correctly declares a two dimensional integer array with 3 rows and 4 columns?

a) int arr[3, 4];

b) int arr(3)(4);

c) int arr[3][4];

d) int arr[3-4];

12. In C, a string is internally represented as:

a) A built-in string data type

b) A character array terminated by a null character

c) An integer array

d) A pointer to an integer

13. Which function is used to find the length of a string in C?

a) strcat()

b) strcmp()

c) strlen()

d) strcpy()

14. What does strcmp(str1, str2) return when the two strings are identical?

a) 1

b) -1

c) 0

d) The length of str1

15. Which header file must be included to use functions like strcpy(), strcat(), and strcmp()?

a) stdio.h

b) stdlib.h

c) string.h

d) math.h

Answer Key

1-b, 2-c, 3-c, 4-b, 5-c, 6-c, 7-b, 8-b, 9-c, 10-c, 11-c, 12-b, 13-c, 14-c, 15-c

Broad / Descriptive Questions (10)

1. Explain the if-else statement with syntax and a suitable example. How does an else-if ladder differ from a set of independent if statements?

2. Describe the switch case statement with its syntax. Explain the role of the break and default keywords, and state two limitations of switch compared to if-else.

3. Differentiate between entry controlled and exit controlled loops. Explain why do-while always executes its body at least once, while while may not.

4. Compare the while, do-while, and for loop structures with respect to their syntax and typical use cases, giving one example program for each.

5. Explain the break and continue statements with the help of example programs, clearly showing how their effects on loop execution differ.

6. What is a goto statement? Explain conditional and unconditional goto with examples, and discuss why the use of goto is generally discouraged in structured programming.

7. What are the advantages of using arrays (subscripted variables) over declaring separate variables for each value? Explain with an example.

8. Explain the declaration, initialization, and element access of one dimensional and two dimensional arrays in C, with suitable example programs.

9. What is a character array? Explain how strings are declared, initialized, and terminated in C, and how a character array differs from a numeric array.

10. Explain any four string handling functions available in the string.h library. Write a C program to concatenate two strings and compare them using strcmp().