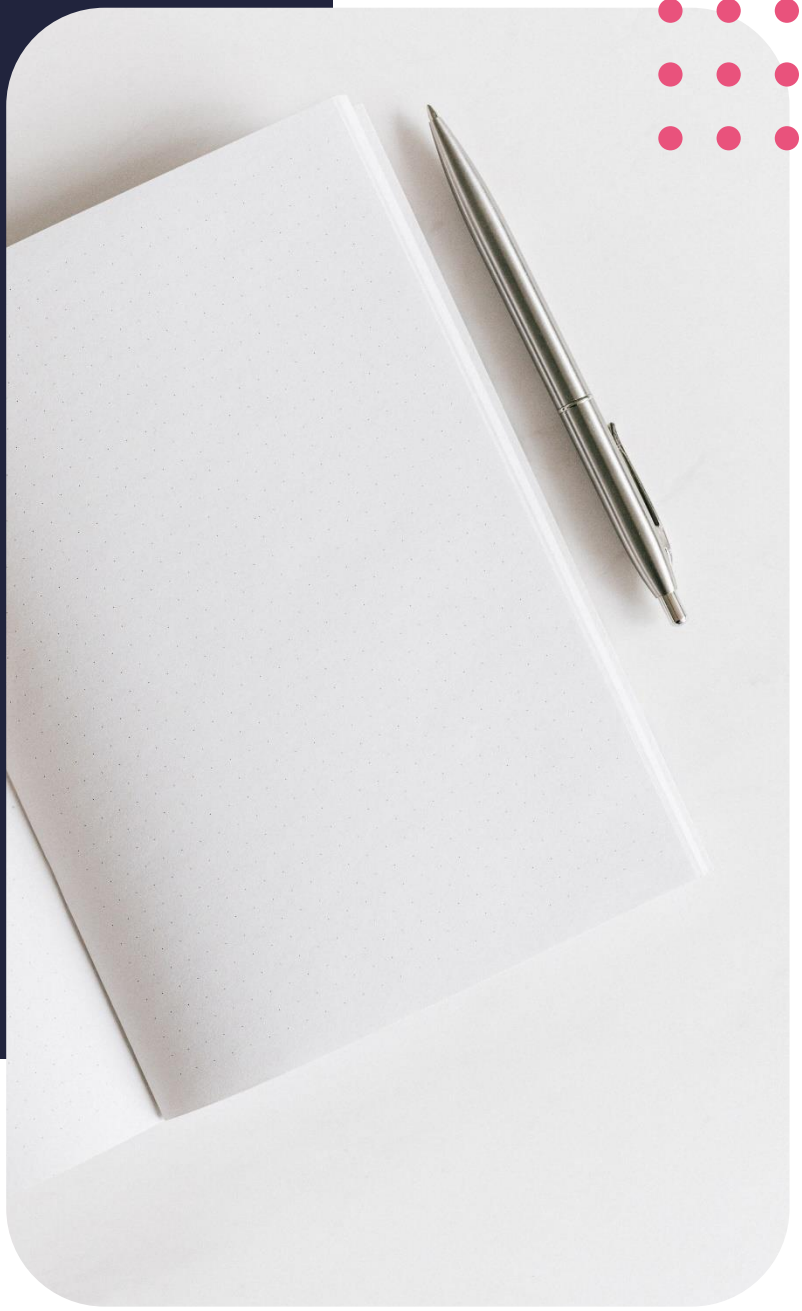
Diploma in Computer Science

Decision making

Contents



Lesson outcomes

By the end of the lesson, you should be able to:

- Understand the concept of decision making in C
- Learn how to create a repetitive block of code
- Appreciate the different ways of achieving repetition
- Explore the methods of counting automatically

Smarter code

As you may have noticed already, it's not always that a program is straightforward and does only one thing all the time. Sometimes you need to perform certain actions depending on what the program receives as input. The program needs to know what to do when presented with different sets of information, otherwise we would have to have many different sets of programs for different functions, and that would make the computer's abilities pretty limited as we would have to "think" for it whenever we want to perform certain sets of calculations and processes. Decisions are what your phone to calculate the shortest route to a destination when you are using GPS navigation. This is also how your camera knows that you are capturing a night shot and adjusts camera settings accordingly. Of course, it's a lot more complex than the examples that we are going to look at in this lesson, but the basic concept is exactly the same. Let's jump in!

Uncertain actions in programs

Like we just mentioned, it's not always that your program does the exact same thing every single time. Sometimes you need it to choose between a number of blocks of code to make it do the appropriate action for the data that it is presented with. Let's face it, it's not always that you know the data that you are going to be working with, so there are very few times when you know what to expect. There are also critical functions such as error handling that rely on the decision-making capabilities of programming languages.

You need a flowchart!

Our flowcharts for the programs that we have been writing and running were fairly simple. They had simple flowcharts that barely used up half an A4 sheet. You could get away with writing such a program right off the bat without any pseudocode, and still come up with a well written program that is very efficient and easy to read without having to do the flowchart first. Now imagine writing a program with 300 if statements. You would probably get lost in the first 20 and give up on trying to understand the code.

The more complex your code becomes, the more it becomes necessary to plan your code. A flowchart quickly becomes a valuable friend and will help you as well to understand the flow of your code and avoid errors.

A little later in the lesson, we are going to look at more complex control structures that involve a lot of branches. That is where you will appreciate the importance of flowcharts, as things escalate pretty quickly and tangle you in code.

Decisions

Decisions. They run the world, from the moment you wake up, you will be making decisions. Deciding to respond to the alarm and wake up, deciding what to wear, deciding that you still have enough fuel in your car and don't need to stop at the petrol station. Deciding it's time to leave for work. The least is endless.

There's one common thing in all these decisions and all those that we did not mention. There is a set of conditions which we evaluate and choose the appropriate course of action. If we think in computing terms, this is exactly what a computer would do. A condition, or a set of conditions are provided in the code, and the computer evaluates these conditions. The computer then decides the order of execution based on the outcome of the evaluation of the conditions.

Decisions are handles in C programming in 4 ways, namely the

- If statement
- Switch statement
- Conditional operator statement (? : operator) and the
- Goto statement

Each of these have at least one condition that they evaluate in order to decide which block of code is to be executed. Typically, you provide the computer with two blocks of code: the first block of code is executed only if the condition is met, while the other one is executed if the condition is not met. At the very basic level, all decision statements are exactly the same.

There are a number of conditions that you can evaluate in decision statements. This table summarises the arithmetic evaluations that you can use in decision making operations.

Operator	**Meaning**	**Example**
==	Strict equal to	a === 5
!=	Strict not equal to	a !== 5
>	Greater than	a > 5
>=	Greater than or equal	a >= 5
<	Less than	a < 5
<=	Less than or equal	a <= 5

Sometimes evaluation of 1 comparison is not enough to determine if the computer can go ahead and execute a block of code. This is where logical operators come in. I'm sure you still remember these from our lesson on operators. These allow you to base the execution on multiple conditions. Think of how you would decide to wear shorts. If it is hot and calm, then it's a very nice day to wear shorts. If it is hot but very windy, then shorts won't be such a great idea because the wind will beat the living daylights out of your legs. You would probably choose to wear sweatpants instead.

Back to actual code, you can use the exact same reasoning in your code to evaluate multiple conditions at once. You can also use multiple logical operators of either the same type or different types, just be careful not to get lost in your own logic! Another point to note is that operator precedence rules apply here as well, so your evaluations need to be in the right order.

Simple conditions

The simplest form of a conditional execution statement has a statement that is executed when the condition is met, and if the condition is not met, then it just carries on with the program as if nothing happened. This is useful when you want to modify a value based on certain conditions, and that modification is only done explicitly when the condition is met and not any other time.

if statement

This is naturally the place to start when talking about decisions. If it still seems a bit challenging to grasp, think of it this way. If you are hungry, then you go to the kitchen and grab a snack. If no, then you carry on with what you were doing. That is actually what the basic syntax of the if statement looks like! The basic syntax looks like this

```
if(condition)
{
    Statement executed when condition is met;
}
    statement executed when condition is not met;
```

Here, if you are going to be using only one condition, then you can get away with not including curly brackets and your syntax would look like this

```
if(a<5)
    printf("This number is too large");
```

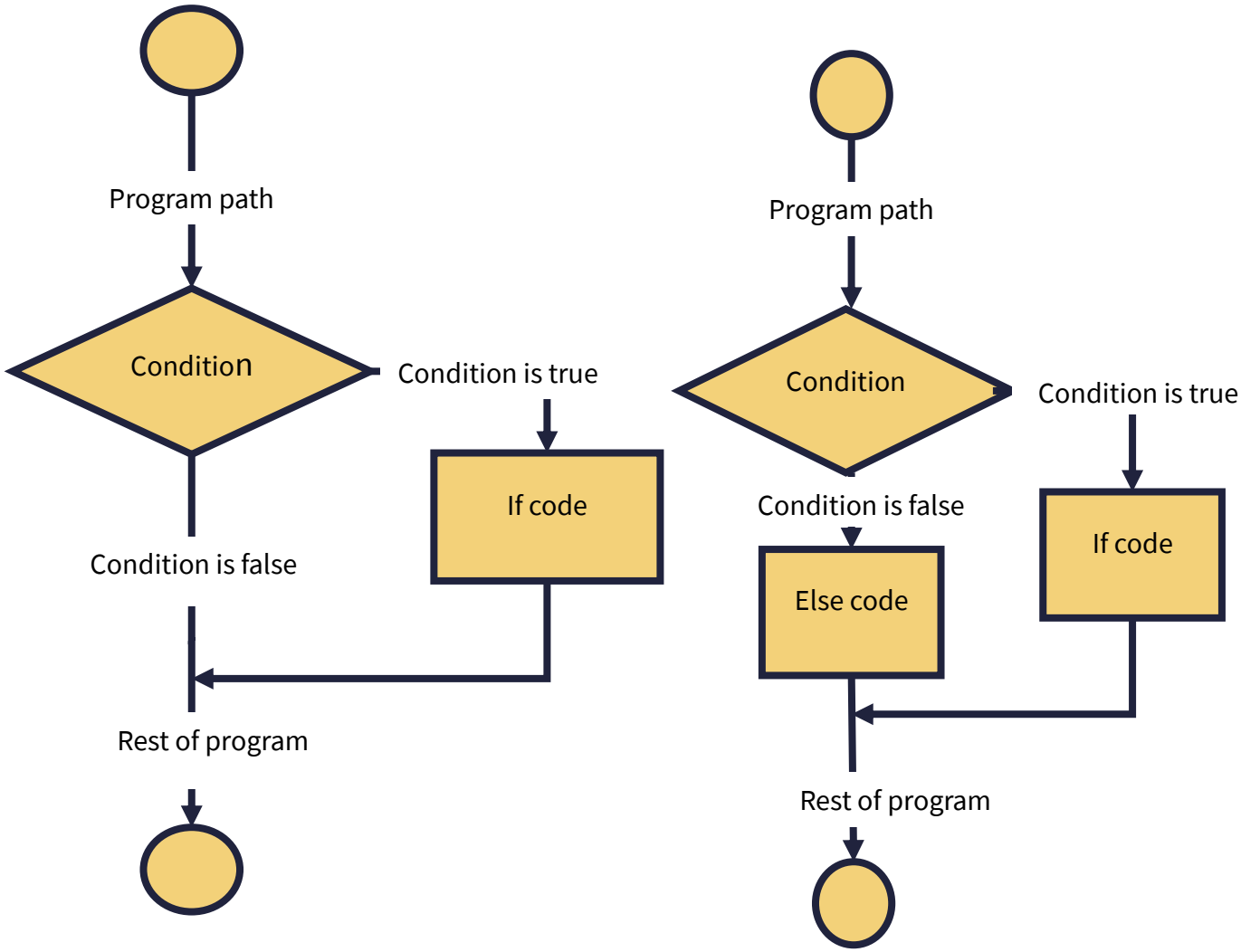
This code will execute just fine as it is and is syntactically and semantically correct.

Another point to note is that the condition being evaluated must be a Boolean condition. This means the condition can only be either true or false. If we look at the code snippet in the snippet that we just looked at, we will see what this means. The value of a can only be either greater than 10 or smaller than 5. The values that the program will be interested in are values that are greater than 5. Anything else does not satisfy the condition and so is considered false. It's only yes or no here, there can't be "maybe" or "somewhat". This reflects the quality of code that we discussed in earlier lessons known as ambiguity. Just out of interest, there is a branch of computer science known as fuzzy logic that deals with that is somewhat true and somewhat false, but that is content for a whole course! Back to our if statement, there ais one other thing that you need to be wary of. When creating expressions. What do you do when you want to test if a variable is of a specific value, for example checking if a=5. You would think that you would type if (a=5) right. Well, actually, no. That statement would actually assign the value 5 to a, and so your code would not run as intended. Instead, you need to use the comparison operator ==, so that the computer knows it's supposed to compare a and 5 instead of assigning the value to a. It's pretty easy to ruin a whole block of code with just one little operator.

If-else statement

You can expand the if statement, so that instead of just executing a block of code when the condition is true then carrying on with the program, you have an if statement that executes a certain block of code when the condition is met, then another block of code if the condition is not met, then carries on with the program.it does not look like it's any different from the first way in which we did it, but it goes a long way in removing ambiguity from your code, as we shall see as we progress with this lesson.

This flowchart snippet does a better job at explaining this concept than what words alone or code would.



else-if statement

Another way of using the if statement is to stack statements so that there are a number of alternate statements to be executed. A classic example that demonstrates this concept is a program that calculates the appropriate symbol given a mark. Let's say a school uses a grading system that goes like so

A=80-100

B=70-79

C=60-69

D=50-59

E=40-49

F=0-40. Given a mark, say 89, then the program will check if the mark is below 40, since it's not, the program will check again if it is between 40 and 49. The condition is still not satisfied so the program moves to the next if block, and this goes on and on till the mark finds its place and returns the letter A and voila! A symbol for our bright, high flying student!

It’s worth noting here, though, that when your if statements keep going on and on, it would be better to use the case statement. We shall see why a bit later in this lesson.

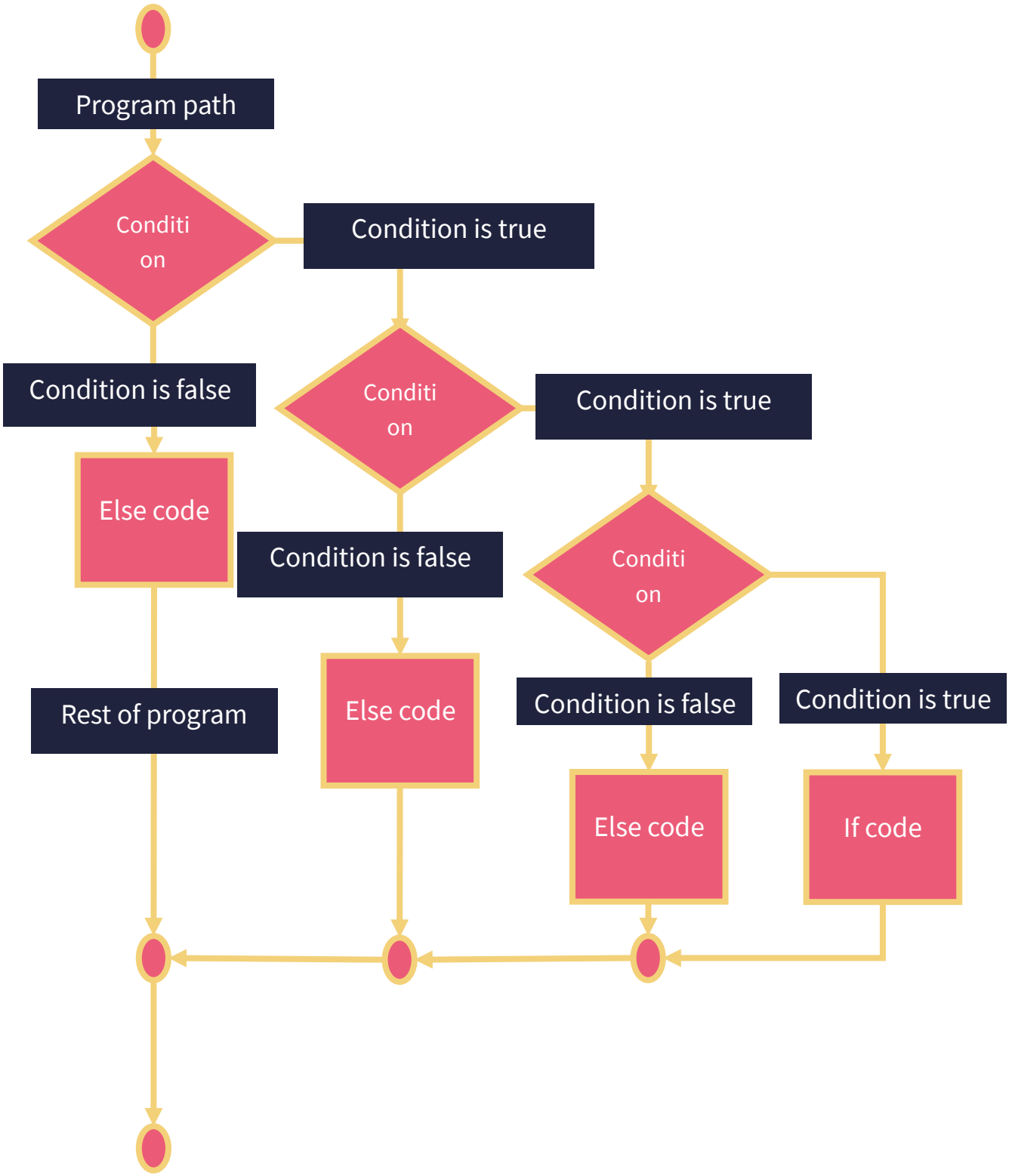
There are a number of points to note when using if statements. A if statement can be followed by an optional else if...else statement, which is very useful to test various conditions.

- An if can have zero or one else's and it must come after an else if.
- An if can have zero or more else…if's and they must come before the else.
- Once an else…if succeeds, none of the remaining else…if's or else's will be tested.

The if else statement is a very powerful tool for conditional execution, and one of the most straightforward concepts in programming. Here’s a pro tip: if you find yourself somewhat confused by an if statement, just read it as though you were reading an actual English sentence.

Nested if statement

A nested if statement is one that exists within another if or else statement. This means that if a condition returns true (or false in the case that the next if is within an else statement) it triggers another evaluation. It is different from the if-else if statement in that instead of existing at the same level, the if statements exist one level lower at each stage and will be evaluated backwards till, we return to the initial level. Sounds like a mouthful? Let’s take a look at a flowchart that demonstrates this.


Program path
Conditi
on
Condition is true
Condition is false
Else code
Rest of program
Conditi
on
Condition is true
Condition is false
Else code
Conditi
on
Condition is false
Else code
Condition is true
If code

When executing an if statement, you will immediately pick up that there is a problem if there's more than one block of code executed after the evaluation. This bug is pretty hard to introduce in your code since the if statement is so straightforward, but it's a possibility, especially if you are using nested if statements. Be sure to take note of the example that we have lined up in today's demo.

Switch statement

We mentioned earlier in the lesson that there is a better way of dealing with a lot of conditions. Well, this is it. The switch statement lets the computer choose a block of code based on the condition among a bunch of conditions that satisfies the comparison. The easiest way to visualise it is to think of a tree. It has a lot of branches that each go their separate way. Similarly, each condition has each own code path that it follows, which is completely different from all of the other branches. The more appropriate name for this type of structure is a selection structure.

Multiple options

The switch statement is typically used when you have a number of options

Each of the blocks of code that can be executed is called a case. Each case has its own unique identifying constant. This constant will be a result of the expression evaluation in the switch statement.

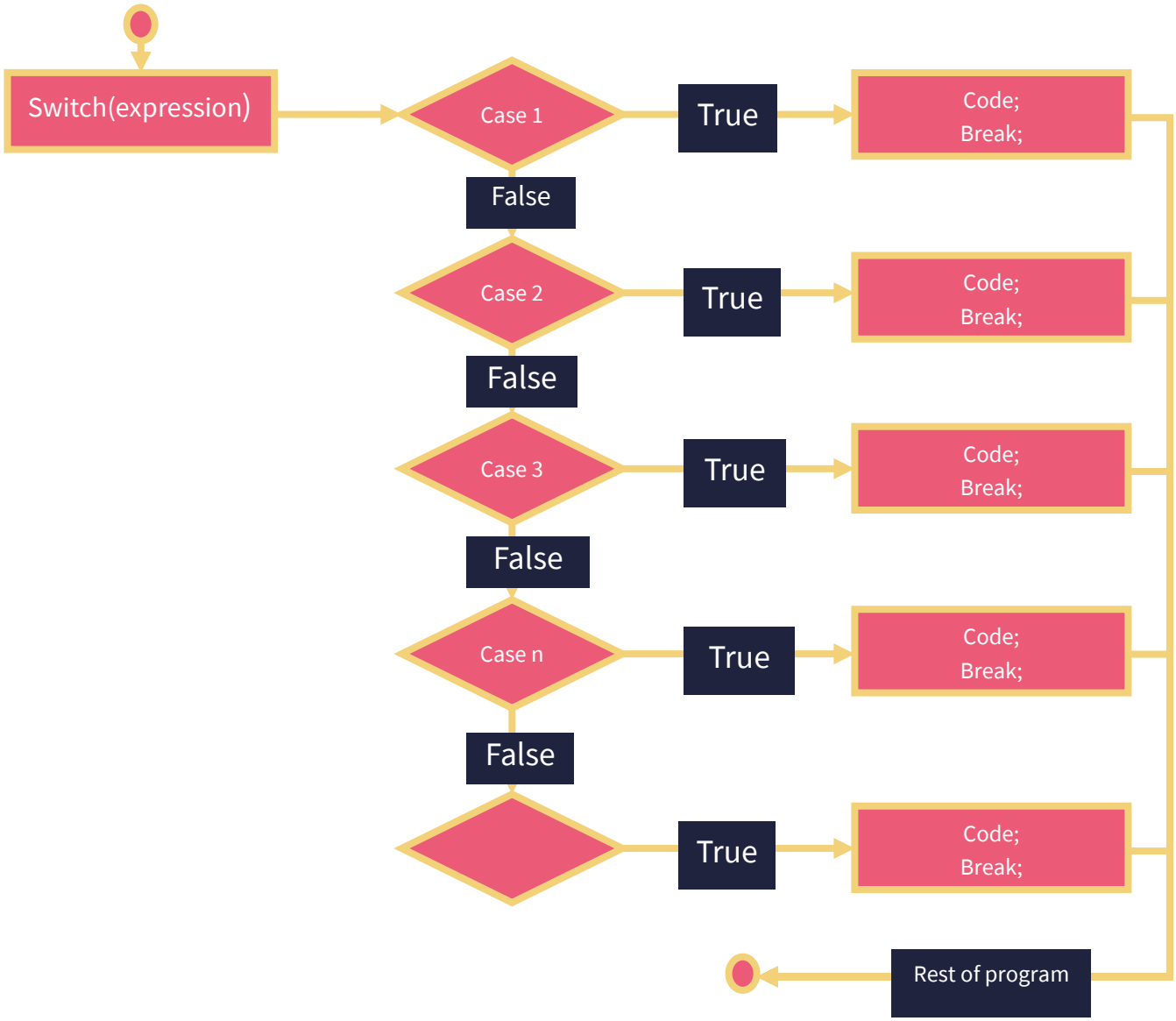
Syntax

There are a few rules that you need to need to stick to when dealing with switch statements.

- The expression in the switch statement can be a variable or an expression – but you need to make sure that whatever results from the evaluation is either an integer or a character.
- The case label must be a literal or a variable declared to be const. This one is probably the strictest of them all; you can't have case labels with the usual variables, strings, floating point literals, operations, or function calls. It has to, HAS TO be a constant.
- You can have as many cases as you like, but you can't have any duplicates. Switch statements can also be nested within other switch statements, just like we did with if statements.
- The optional default case is executed when none of the cases match the expression evaluation.

The break statement is used to break the flow of control when the computer encounters a matching case and executes the relevant code. If you leave it out, then the computer will just go ahead and execute the rest of the code that follows the selected case, since it won't know where to stop, and that defeats the purpose of having a switch statement in the first place. The program will simply execute until it encounters a break, and if there isn't any, then ALL the blocks of code in cases after the selected case will be executed, then the computer will move to the rest of the program. Yikes!

A simple fun experiment that you can do with the switch statement is to recreate the USSD menu on your phone, that it, that menu you get after dialling something like *147# and you get options for checking your balance, buying airtime and the like.



The syntax of a switch statement looks like this

```
switch(expression)
{
        case value-1:
                code;
                break;
        case value-2:
                code;
                Break;
        case value-n:
                code;
                break;
        default:
                code;
                break;
}
```

The expression, as mentioned, can only evaluate to an integer or a character. Anything else, and you will be cordially inviting errors in your code! The result of the expression evaluation is then compared to the case values. If the case value that matches the expression value is encountered, the block of code under that case

value is executed, until the computer encounters a break, then exits the entire switch structure and carries on execution of the rest of the program.

The magic in this structure happens in the switch expression. The example we looked at above is the easiest and the closest to home when trying to understand how the switch statement works. When you dial the USSD code, you are presented with options that look something like this

1. Self service
2. Check balance
3. Buy airtime
4. SIM swap
5. Port my number
6. Other services

The menu typically requires you to enter the number corresponding to the service that you want to use. This number is what we are interested in. remember we said the switch statement accepts integers and characters. Let's say the user chooses option 2. The computer then goes through the cases until it encounters option 2, then executes the code that is under that option. If for example, the user gives input that doesn't have a corresponding case, the computer will run the code that is under the default case, then exits the switch structure. There should always be a way for the computer to exit the switch structure, either by finding a matching case or running the default option.

You may be wondering what kind of expressions you can use in the switch structure. Well, it can be an actual mathematical or logical expression or a variable, or a constant, whatever rocks your boat. The key here is just to make sure that whatever you put in there returns either an integer or a character.

Conditional operators

Time for a bit of retrospect! Remember that conditional operator, the ternary operator that we looked at when we were discussing operators? Turns out, it can be used to create short expressions that work in pretty much the same way as if-else statements

The ternary operator, as we saw on the lesson on operators, takes 3 arguments

The first argument is a comparison argument. This is what the computer will compare.

The second is the result that is adopted if the comparison is true

The third is the value that is adopted if the comparison is false.

This is shorthand of sorts for the if statement, in fact, you can even read it the same way that you would read an if statement!

Let's look at this code snippet and see what we are trying to get at here.

```
int a = 4, b = 5, c = 0;
if (a < b) {
    c = a;
}
else {
    c = b;
}

printf("%d is smaller", c);
```

If we run this code, we will get the output “4 is smaller” That’s as simple as it can get, right? What if I told you that there was a better way of doing this?! We can make our code lose a lot of weight using the conditional operator.

Here’s the code snippet after we rewrite it using the ternary operator.

```
int a = 4, b = 5, c = 0;
c = (a < b) ? a : b;
printf("%d is smaller", c);
```

You may be wondering what just happened there. Well, exactly the same thing that happened with the if statement! What that skinny little line of code does is look at the values on the right, then assign the value that meets the condition in brackets to C! We shaved off a whopping 5 lines of code by just using the conditional operator! This saves you a ton of typing and also makes the code a whole lot more readable, as there are a lot less lines of code to keep track of while trying to make sense of the operation. It is worth noting here, from the book of rules and laws, that the arguments value_if_true and value_if_false must be of the same type, and they must be simple expressions rather than full statements.

Wait, there’s more good news! The ternary operator can also be used in nested statements, just like we did in if-else statements. Let’s take a look at this code snippet, which uses the if else statement.

```
int a = 4, b = 5, c=0;
if (a == 4) {
    if (b == 5) {
        c = 3;
    } else {
        c = 5;
    }
} else {
    c = 0;
}
printf ("%d\n", c);
```

This looks pretty confusing and rather long.

If we rewrite the code using ternary operators, well get this short and simpler snippet.

```
int a = 4, b = 5, c=0;
c = (a == 4 ? (b == 5 ? 3 : 5) : 0);
printf ("%d\n", c);
```

We now have a shorter code snippet that’s simple, to the point and a lot easier to read. It’s a good idea to give yourself time to play around with ternary operators, coming up with various problems that you can solve using the ternary operator. It really goes a long way in helping you write shorter code, which makes you write programs a lot faster.

We’ve seen that the if-else statement and the ternary operator share a lot of similarities, but there are a number of ways in which the two differ.

- The conditional operator is a single statement while the if else statement is a block of code; this one goes without saying.
- The conditional operator can be used to assign a value to a variable, but you can’t use the if else statement for that, at least not directly, since you can do it anyway in the block of code.
- As superb as the conditional operator is, it’s really not your best choice when you have multiple

conditions. The if else statement works better here.

- The nested ternary operator can quickly become complex and will be very hard to debug. If it's a complex operation, rather stick to the if else statement.

Conclusion

On that we have come to the end of our lesson. It's been really interesting and contains some of the most important concepts that you will use a lot in programming. We discussed the ways in which computers make decisions, and the various methods of making decisions in a program. We also discussed the importance of flowcharts as programs become more and more complex. In our next lesson, we will look at the ways in which you can get the computer to do repetitive tasks in the form of loops and iteration. We will also look at the traps which you can easily fall into when using loops. Gear up as we are going to look at a key concept in computer science and programming, known as recursion. We will look at how this concept is used in complex operations. Just a forewarning, it's usually challenging to a lot of students, so try to read ahead a bit.

References

Edpresso Team (2019). How to use the switch statement in C. [online] Educative: Interactive Courses for Software Developers. Available at: https://www.educative.io/edpresso/how-to-use-the-switch-statement-in-c

corob-msft (2020). switch Statement (C). [online] Microsoft.com. Available at: https://docs.microsoft.com/en-us/cpp/c-language/switch-statement-c?view=msvc-160

examtray (2019). Last Minute C Programming Conditional Operators or Statements Tutorial. [online] ExamTray. Available at: https://www.examtray.com/tutorials/last-minute-c-programming-conditional-operators-or-statements-tutorial

freeCodeCamp.org (2020). Ternary Operator in C Explained. [online] freeCodeCamp.org. Available at: https://www.freecodecamp.org/news/c-ternary-operator/

fresh2refresh.com (2020). Conditional Operators in C | C Operators and Expressions | Fresh2Refresh. [online] fresh2refresh.com. Available at: https://fresh2refresh.com/c-programming/c-operators-expressions/c-conditional-operators

Fsu.edu. (2020). Control Structures. [online] Available at: https://www.cs.fsu.edu/~myers/c++/notes/control1.html

Fsu.edu. (2020). Control Structures. [online] Available at: https://www.cs.fsu.edu/~myers/c++/notes/control1.html

Khan Academy. (2015). If/Else - Part 2 | Logic and if Statements | Intro to JS: Drawing & Animation | Computer programming | Computing | Khan Academy. [online] Available at: https://www.khanacademy.org/computing/computer-programming/programming/logic-if-statements/pt/ifelse-part-2

Rungta, K. (2020). C Conditional Statement: IF, IF Else and Nested IF Else with Example. [online] Guru99.com. Available at: https://www.guru99.com/c-if-else-statement.html

Studytonight.com. (2020). C Language Decision Making - if, else and else if statemente=s | Studytonight. [online] Available at: https://www.studytonight.com/c/decision-making-in-c.php

www.javatpoint.com. (2011). Conditional Operator in C - javatpoint. [online] Available at: https://www.javatpoint.com/conditional-operator-in-c