COE 251

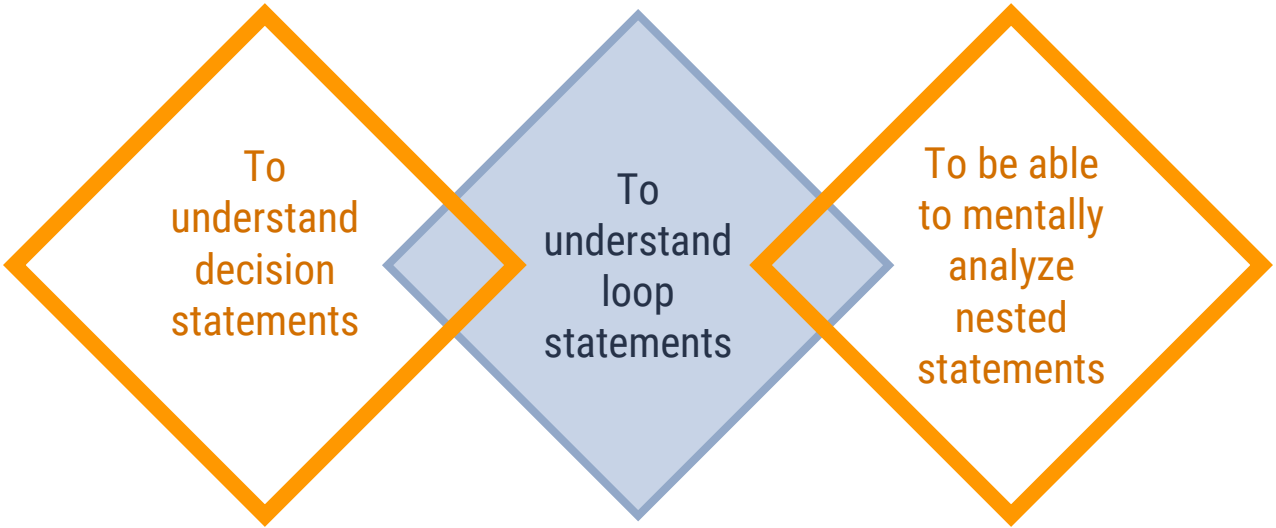
FLOW OF CONTROL

Dr. Eliel Keelson

OUTLINE

- DECISIONS
- LOOPS
- NESTED STATEMENTS

INTENDED LEARNING OUTCOMES (ILOs)



FLOW OF CONTROL

FLOW OF CONTROL

- Most programs (like many humans) decide what to do in response to changing circumstances.
- For example selecting a particular option/choice out of a number of options presented or performing an activity a number of times depending on the prevailing condition.

FLOW OF CONTROL

- Program statements that help in making a choice or a repetitive action are called **control statements**.

- There are two major categories of control statements: **decisions** and **loops**.

1

DECISION STATEMENTS

DECISION STATEMENTS

- Decision statements are control statements that enable the program select an option from a number of options presented.
- This selection is made based on a prevailing condition which should be true as at the time of selection.
- This is very similar to decisions that humans take at certain times. Their final decision are guided by conditions.

DECISION STATEMENTS

- For example one would decide to eat when hungry and decide not to do so when not hungry.

- Or probably selecting a particular meal to eat depending on the time of the day.

DECISION STATEMENTS

- There are two kinds of decision statements that are used in programming with C.
- They are:

1. if else statements and
2. switch case statements

DECISION STATEMENTS: if else Statements

- The general syntax for the if else statement is:

```
if (condition)
{
        statements;    // code to execute if condition is true
}
else
{
        statements;    // code to execute if condition is false
}
```

DECISION STATEMENTS: if else Statements

- Bearing in mind that C is a high level language, the if else syntax can be easily read as: if the condition is true execute the statements under the if, else if the condition is false execute the statements under the else.

DECISION STATEMENTS: if else Statements

- Lets take an example:

```
int x = 20;
if (x % 2 == 0)
{
        printf("The value is even \n");
}
else
{
        printf("The value is odd \n");
}
```

DECISION STATEMENTS: if else Statements

- The code in the previous slide can be easily read in plain English as: if x modulus 2 is equivalent to zero print out “The value is even” else print out “The value is odd”.
- This is a simple demonstration of selecting one out of two options depending on the condition.

DECISION STATEMENTS: if else Statements

- The previous program could have been written to accept a user's input rather than having the programmer stating statically that the value of x is 20.
- This is demonstrated in the next slide

DECISION STATEMENTS: if else Statements

```
int x;
printf(“Please enter an integer value \n”);
scanf(“%d”, &x);
if (x%2 == 0)
{
        printf(“%d is even \n”, x);
}
else
{
        printf(“%d is odd \n”, x);
}
```

DECISION STATEMENTS: if else Statements

- The previous programs present only two options; what to do when the statement is true and what to do when the statement is false.
- However, there are situations when there are more than two options.
- In such situations there is the introduction of an else if in addition with a condition as shown in the next slide

DECISION STATEMENTS: if else Statements

```
if (conditionA)
{
        statements;    // code to execute if conditionA is true
}
else if (conditionB)
{
        statements;    // code to execute if conditionB is true
}
else
{
        statements;    // code to execute if all conditions above are false
}
```

DECISION STATEMENTS: if else Statements

- From the previous slide, it is obvious that this syntax presents the compiler with 3 options and only one would be selected depending on which condition is true.
- Since C executes procedurally, starting from the top, the very first condition that the compiler sees to be true would be selected its code executed.
- Even if there are two true conditions.

DECISION STATEMENTS: if else Statements

```
int numb;
printf("Please enter an integer value \n");
scanf("%d", &numb);
if (numb < 0)
{
        printf("%d is negative \n", numb);
}
else if (numb > 0)
{
        printf("%d is positive \n", numb);
}
else
{
        printf("%d is zero \n", numb);
}
```

DECISION STATEMENTS: if else Statements

- From the example above, we realize that the program tells the user whether the value he/she enters is positive, negative or zero (3 options).
- So to increase the number of options one only needs to include the following as many times as he/she wants:

```
else if (condition)
{
        statements;
}
```

DECISION STATEMENTS: if else Statements

- From the examples above, it is obvious that the else statement (the part at the far bottom) is the default statement. Hence in case all the above conditions are false the else statement is executed.
- However, it is important to note that the else statement is not **mandatory** and as such may not always appear when a programmer writes an if else statement.

DECISION STATEMENTS: switch case Statements

- Another decision statement in C is the switch case statement.
- It is very similar to the if else statement. It could be used in places where there are many if else statements as in so many multiple if options
- However, in the switch case, the exact values that match an option are used in making a decision.
- The following slide shows its general syntax.

DECISION STATEMENTS: switch case Statements

```
switch (expression)
{
        case I: statements;
                break;
        case II: statements;
                break;
          .
          .
          .
        case N: statements;
                break;
        default: statements;
                break;
}
```

DECISION STATEMENTS: switch case Statements

- In the switch, the statements under the case that matches the expression are executed.
- The matched case determines where we start executing from and the break tells us where we end.

DECISION STATEMENTS: switch case Statements

- The break means “**stop and exit**”, it is synonymous to the car’s **brake** which causes the car to stop.
- So without the break we would keep executing all the statements downwards under the matched case until we meet a break or come to the end of the switch.
- In cases where none of the cases match the expression, the statements under the default are executed.

DECISION STATEMENTS: switch case Statements

```
int choice;
printf(“Are you a Ghanaian \? \n”);
printf (“Enter 10 for yes of 20 for no \n”);
scanf(“%d”, &choice);
switch(a)
{
        case 10: printf (“Ghanaians are hospitable people \n”);
                 break;
        case 20: printf (“I don’t know much about your country \n”);
                 break;
        default  : printf (“You did not follow the instruction \n”);
}
```

DECISION STATEMENTS: switch case Statements

- The code in the previous slide aims at finding out if the user is a Ghanaian. The user is asked to enter 10 for yes and 20 for no.
- If the user's answer is 10, case 10 would execute and display just "Ghanaians are hospitable people" and then stop and exit the switch (because of the break that follows the printf statement)

DECISION STATEMENTS: switch case Statements

- However, if the user's answer is 20, case 20 would execute and display just "I don't know much about your country" and then stop and exit the switch (because of the break that follows the printf statement).
- Also if the user's answer is neither 10 nor 20 the default case would be executed

2

LOOP STATEMENTS

LOOP STATEMENTS

- Loop statements enable certain programming operations or sections of code to be executed repeatedly as long as a prevailing condition exists or is true.
- Each loop/cycle of execution is seen as an iteration.

LOOP STATEMENTS

- There are three kinds of decision statements that are used in programming with C.
- They are:

1. while loop statements
2. do while loop statements and
3. for loop statements

LOOP STATEMENTS – while loop

- The while loop is used to iterate (loop) some specific set of programming instructions a number of times based on a specified condition.
- The code would keep executing repeatedly until the condition stated becomes false.

LOOP STATEMENTS – while loop

- The general syntax for the while loop is:

```
while(condition)
{
        statements;
        //statements to be executed as long as condition is true
}
```

LOOP STATEMENTS – while loop

- Bearing in mind that C is a high level language, the while loop syntax can be easily read as: while the condition is true execute the statements under the while.

LOOP STATEMENTS – while loop

- Lets take an example:

```
int q = 5;
while(q<8)
{
        printf("%d\t", q);
        q++;     /* increase the value of q by 1*/
}
```

LOOP STATEMENTS – while loop

- From the code on the previous slide, initially q has a value of 5 and that makes the condition true (i.e. q<8) so the while loop executes the statement within it.
- The statements within the loop print out the current value of q which is 5 and then after increments the value of q by 1(i.e. q ++) therefore q would now have a new value of 6 in computer memory.

LOOP STATEMENTS – while loop

- Since it is a loop we would go back and check the condition if it is still true.
- If true we keep executing the statements until the condition turns false. So the result of the above code is

```
5   6   7
```

LOOP STATEMENTS – while loop

- Lets take another example:

```
int b = 10;
while(b>=7)
{
        printf("%d\t", b);
        b--; /* decrease the value of b by 1 */
 }
```

LOOP STATEMENTS – while loop

- This code is very similar to the previously explained code.
- If true we keep executing the statements until the condition turns false, the output of the code would be

```
10  9  8  7
```

LOOP STATEMENTS – while loop

- Note that the position of the statements is very important.
- For the next example we switch the statements around. Let’s see the output generated

LOOP STATEMENTS – while loop

- Lets take another example:

```
int b = 10;
while(b>=7)
{
        b--; /* decrease the value of b by 1 */
        printf("%d\t", b);
 }
```

LOOP STATEMENTS – while loop

- By virtue of the position of the statements, despite the similarity to the previously executed code, this code would result in the following output

```
9   8   7   6
```

LOOP STATEMENTS – do while loop

- The do while loop is very much similar to the while loop
- However, in the do while loop the statements are executed first before the condition is checked.
- As such, in the do while loop, we are always guaranteed of at least **one iteration**.

LOOP STATEMENTS – do while loop

- The general syntax for the do while loop is:

```
do
{
        statements;
        //statements to be executed
}
while(condition); // take note there is a semicolon
```

LOOP STATEMENTS – do while loop

- So in the very first instance whether the condition is true or false we go ahead and do the statements before checking the condition.
- And we only continue to do the statements again if the condition is true.

LOOP STATEMENTS – do while loop

- Lets take an example:

```
int m = 1;
do
{
        printf("%d\t", m);
        m++;
}
while(m<=5);
```

LOOP STATEMENTS – do while loop

- From the previous slide, the code start by first initializing the value of m to 1.
- We then go to the do while statement to execute the body of statements first before checking the condition.
- After the first execution we check the condition and keep executing until the condition becomes false.
- So the output of the code would be | 1 2 3 4 5 |

LOOP STATEMENTS – do while loop

- Lets take another example:

```
int m=1;
do
{
        printf("%d\t", m);
        m++;
}
while(m>=5);
```

LOOP STATEMENTS – do while loop

- From the previous slide, it is obvious that the condition is false from the beginning.
- However, since in the do while we ought to execute the body of statements before we check the condition the output of the code would be 1

LOOP STATEMENTS – for loop

- One of the most used loops in C is the for loop.
- It is often used because it is easy to specify how many times the statements should be looped.
- It is also the preferred choice when iterating through arrays.

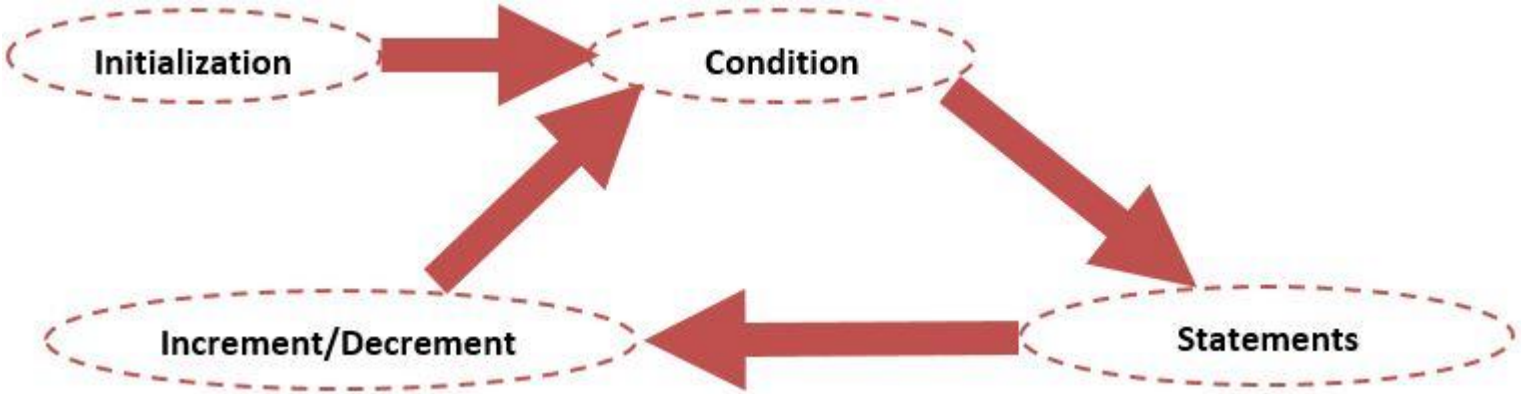
LOOP STATEMENTS – for loop

- The general syntax for the for loop is:

```
for(initialization; condition; increment/decrement)
{
        statements;
}
```

LOOP STATEMENTS – for loop

- The general algorithm for executing the for loop is shown below



LOOP STATEMENTS – for loop

- From the algorithm, it is clear that the initialization is only done once.
- Also after the condition is evaluated to be true, the statements are executed and after the increment/decrement is done (not the other way round!).
- The for loop would keep iterating as long as the condition is true

LOOP STATEMENTS – for loop

- Lets take an example:

```
int a;
for(a=0; a<4; a++)
{
        printf(“%d\t”, a);
}
```

LOOP STATEMENTS – for loop

- Going by the earlier explained algorithm, the for loop for the example on the previous slide would have the following output.

```
0 1 2 3
```

LOOP STATEMENTS – for loop

- Lets take another example:

```
int c = 0;
for(c=1; c<10; c*=2)
{
        printf(“%d\t”, c);
}
```

LOOP STATEMENTS – for loop

- Once again by the earlier explained algorithm, the for loop for the example on the previous slide would have the following output.

```
1  2  4  8
```

3

RULE OF THE BRACES {}

RULE OF THE BRACES {}

- In C, there is a rule for using braces (curly brackets) which state that all programming constructs or elements which usually have the body of their statements encapsulated/compounded by braces, can choose to ignore writing the braces when their body of statements is made up of just one statement.

RULE OF THE BRACES {}

- In other words, constructs and elements such as decision statements, loops and functions can ignore writing/showing braces when they are made up of only one statement.

RULE OF THE BRACES {}

- The statement they contain may be an ordinary expression statement (statement ending with a semicolon) or even a compound statement (such as flow of control statements and functions).
- As long as there is only one statement, the braces can be ignored

RULE OF THE BRACES {}

- For example

```
for(int c=1; c<10; c*=2)
{
        printf("%d\t", c);
}
```

Can be written as such

```
for(int c=1; c<10; c*=2)
        printf("%d\t", c);
```

RULE OF THE BRACES {}

- In the example above, when such a for statement is written without braces, the compiler would interpret that such a for loop has only one statement in it.
- As such, the first statement that follows it would be considered as a part of it.

RULE OF THE BRACES {}

Another example

```
for(int c=1; c<10; c*=2)
{
        printf("%d\t", c);
}
printf("\nDone");
```

Can be written as such

```
for(int c=1; c<10; c*=2)
        printf("%d\t", c);
printf("\nDone");
```

RULE OF THE BRACES {}

- In the second example above, based on the understanding of this rule, it is obvious that the statement printf(“\nDone”); is not a part of the for loop and as such would not be looped when the condition is satisfied.
- We would apply this rule again when we see how compound statements are nested

4

NESTED STATEMENTS

NESTED STATEMENTS

- A nested statement is simply a compound statement in another compound statement.
- As stated earlier, compound statements are statements which are usually followed by braces (eg. Functions, loops and decision statements)

NESTED STATEMENTS

- So whenever we have a compound statement in another compound statement, we describe the inner statement as a nested statement.

NESTED STATEMENTS - Example

```
int x=10, y=0;
while(x<15)
{
        if(x%2 == 0)
        {
                y +=x;
        }
        else
        {
                y--;
        }
        printf("%d", y);
}
```

NESTED STATEMENTS

- In the example, we can see that an if-else compound statement has been placed inside a while compound statement.
- As such the if-else statement can be described as a nested statement.
- The if-else statement can only be executed when the while condition is true.
- The example above would result in

```
10   9   21   20
```

NESTED STATEMENTS

- From the previous example it is obvious that the if and else portions of the code have only one statement as such the rule of the braces can be once again applied to ignore the braces for them.
- This is depicted in the next slide.

NESTED STATEMENTS - Example

```
int x=10, y=0;
while(x<15)
{
        if(x%2 == 0)
                y +=x;
        else
                y--;
        printf("%d", y);
}
```

NESTED STATEMENTS

- Juxtapose the two and see if you can reckon the application of the rule of braces.

THANKS!

Any questions?

You can find me at

elielkeelson@gmail.com & ekeelson@knust.edu.gh