ALGORITHMS AND STRUCTURED PROGRAM DEVELOPMENT

SDN 150S – Week 2

Lecturer: Mr. Stephen Ekwe

Cape Peninsula University of Technology

creating futures

Outline

01

Algorithms

02

Pseudocode

03

Control Structure

04

Conditional Statement

Algorithms

- Before writing a program to solve a problem, you must have a thorough understanding of the problem and a carefully planned solution approach
- The solution to any computing problem involves executing a series of actions in a specific order
- An algorithm is a procedure for solving a problem in terms of the actions to execute and the order in which these actions should be executed.

Algorithms

- Correctly specifying the order in which the actions should be executed is important.
- Consider a "rise-and-shine algorithm" for a junior executive getting ready for work:

```
1.  Get out of bed,
2.  take off pajamas,
3.  take a shower,
4.  get dressed,
5.  eat breakfast, and
6.  carpool to work.
```

- This gets the executive to work well prepared to make critical decisions

Algorithms

- Suppose the steps are performed in a slightly different order

```
1.  Get out of bed,
2.  take off pajamas,
3.  get dressed,
4.  take a shower,
5.  eat breakfast,
6.  carpool to work.
```

- In this case, our junior executive shows up for work soaking wet
- Specifying the order in which statements should be executed in a computer program is called program control

Pseudocode

- Pseudocode is an informal artificial language
- Helps you develop algorithms before converting them to C
- Helps you “think out” a program before writing it in a programming language
- Computers do not execute pseudocode
- You may type it in any text editor
- Often converting carefully prepared pseudocode to C is as simple as replacing a pseudocode statement with its C equivalent

Pseudocode

- Pseudocode describes the actions and decisions
- Definitions are not executable statements, they're simply messages to the compiler
    - `int i = 0`;
    - Tells the compiler variable `i`'s type, instructs the compiler to reserve space in memory for the variable and initializes it to 0
    - Does not perform an action when the program executes, such as input, output, a calculation or a comparison
- Some programmers do not include definitions in their pseudocode

Algorithms vs Pseudocode

C Program

```c
#include <stdio.h>

int main() {
    int num1, num2, sum;

    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    sum = num1 + num2;
    printf("Sum: %d\n", sum);

    return 0;
}
```

Algorithms

1. Start
2. Declare variables num1, num2, and sum as integers
3. Display "Enter two numbers:"
4. Read num1 and num2 from the user
5. Calculate sum = num1 + num2
6. Display "Sum: " followed by the value of sum
7. End

Pseudocode

```
1. Start
2. Declare num1, num2, and sum as integers
3. Display "Enter two numbers:"
4. Read num1
5. Read num2
6. sum = num1 + num2
7. Display "Sum: " + sum
8. End
```

Control Structure

- Control Structures are the blocks that analyze variables and choose directions in which to go based on given parameters
- Normally, statements in a program execute one after the other in the order in which you write them (sequential execution)
- Some C statements enable you to specify the sequence order by which all statements in your code should be executed (transfer of control)

Control Structure

- Bohm and Jacopini's work in 1966 called structured program theorem, demonstrated that all programs could be written in terms of only three control structures, namely;

Sequence structure:

- A sequence structure contains one or more sub-diagrams, or frames, that execute in sequential order.
- Within each frame, as in the rest of the block diagram, data dependency determines the execution order.
- This means the computer executes C statements one after the other in the order in which they're written

Control Structure

Selection Structures:

- The if single-selection statement selects (performs) an action (or group of actions) only if a condition is true
- The if...else double-selection statement performs one action (or group of actions) if a condition is true and a different action (or group of actions) if the condition is false.
- The switch multiple-selection statement performs one of many different actions, depending on the value of an expression.

Iteration Structures:

- These structures are used to perform tasks repeatedly, such as while, do...while, for loops.

Syntax: if Statement

Syntax

```
if(boolean_expression)
{   /* statement(s) will
execute if the boolean
expression is true */
}
```

Given the Boolean condition the if statement give the following result:

```
#include <stdio.h>
int main ()
{
/* local variable definition */
int a = 10;
/* check the boolean condition using if statement */
if( a < 20 )
{
/* if condition is true then print the following */
printf("a is less than 20\n" );
}
printf("value of a is : %d\n", a);
return 0;
}
```

OUTPUT

```
a is less than 20;
value of a is : 10
```

Syntax: if...Else Statement

Syntax

```
if(boolean_expression)
{ /* statement(s) will
execute if the boolean
expression is true */
}
else
{ /* statement(s) will
execute if the boolean
expression is false */
}
```

Given the Boolean expression the if...else statement give the following result:

```
#include <stdio.h>
int main ()
{
/* local variable definition */
int a = 100;
/* check the boolean condition */
if( a < 20 )
{
/* if condition is true then print the following */
printf("a is less than 20\n" );
}
else
{
/* if condition is false then print the following */
printf("a is not less than 20\n" );
}
printf("value of a is : %d\n", a);
return 0;
}
```

OUTPUT

```
a is not less than 20;
value of a is : 100
```

Syntax: if...Else if...Else Statement

Syntax

```
if(boolean_expression 1)
{   /* Executes when the boolean
expression 1 is true */
}
else if( boolean_expression 2)
{   /* Executes when the boolean
expression 2 is true */
}
else if( boolean_expression 3)
{   /* Executes when the boolean
expression 3 is true */
}
else
{   /* executes when the none of
the above condition is true */
}
```

Given the Boolean expression the if...else if...else statement give the following result:

```
#include <stdio.h>
int main ()
{
/* local variable definition */
int a = 100;
/* check the boolean condition */
if( a == 10 )
{
/* if condition is true then
 print the following */
printf("Value of a is 10\n" );
}
else if( a == 20 )
{
/* if else if condition is true */
printf("Value of a is 20\n" );
}
else if( a == 30 )
{
/* if else if condition is true */
printf("Value of a is 30\n" );
}
else
{
/* if none of the conditions is true */
printf("None of the values is matching\n" );
}
printf("Exact value of a is: %d\n", a );
return 0;
}
```

OUTPUT

```
None of the values is matching
Exact value of a is: 100
```

Syntax: Nested if Statement

Syntax

```
if( boolean_expression 1)
{/* Executes when the boolean
expression 1 is true */
    if(boolean_expression 2)
    {/* Executes when the
    boolean expression 2 is
    true */
    }
}
```

Given the Boolean expression the nested if statement give the following result:

```
#include <stdio.h>
int main ()
{
/* local variable definition */
int a = 100;
int b = 200;
/* check the boolean condition */
if( a == 100 )
{
/* if condition is true then check the following */
if( b == 200 )
{
/* if condition is true then print the following */
printf("Value of a is 100 and b is 200\n" );
}
}
printf("Exact value of a is : %d\n", a );
printf("Exact value of b is : %d\n", b );
return 0;
}
```

OUTPUT

```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

If Statement Exercise

1. Write a C program that checks if a given number is positive, negative, or zero.
2. Write a C program to calculate the grade of a student based on their marks.
3. Write a C program that determines if a given number is even or odd and, if even, whether it is divisible by 3.
4. Write a C program to check whether an input alphabet is vowel or consonant using if else.
5. Write a C program to find maximum between three number using nested if statement.

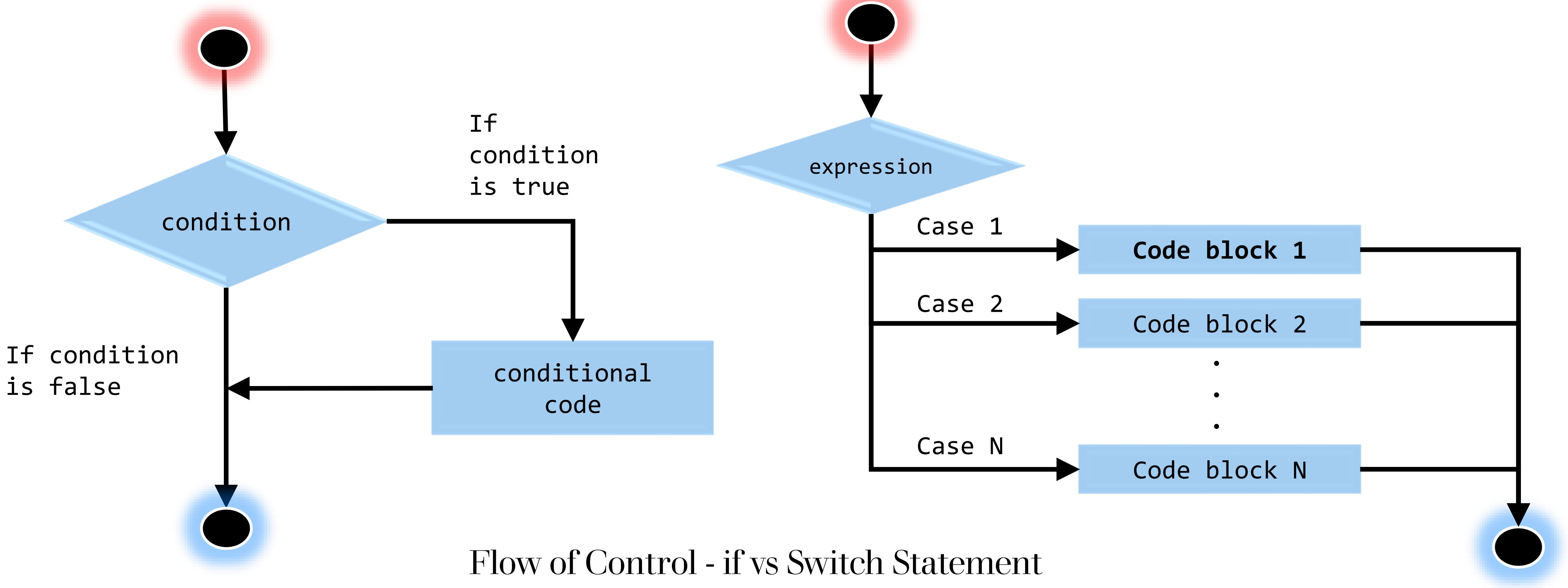
Switch Statement

- A switch statement allows a variable to be tested for equality against a list of values.
- Each value is called a case, and the variable being switched on is checked for each switch case.
- The expression used in a switch statement must have an integral or enumerated type or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch.
- Each case is followed by the value to be compared to and a colon.

Switch Statement

- The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break.
- If no break appears, the flow of control will fall through to subsequent cases until a break is reached.

Switch Statement



Flow of Control - if vs Switch Statement

Syntax: Switch Statement

Syntax

```
switch(expression){
case constant-expression :
        statement(s);
        break; /* optional */
case constant-expression :
        statement(s);
        break; /* optional */
/* you can have any number of
case statements */
default : /* Optional */
        statement(s);
```

The switch statement give the following result:

```
#include <stdio.h>
int main ()
{
/* local variable definition */
char grade = ‘A';
switch(grade)
{
case 'A' :
printf("Excellent!\n" );
break;
case 'B' :
case 'C' :
printf("Well done\n" );
break;
case 'D' :
printf("You passed\n" );
break;
case 'F' :
printf("Better try again\n" );
break;
default :
printf("Invalid grade\n" );
}
printf("Your grade is %c\n", grade );
return 0;
}
```

OUTPUT

```
Excellent!
Your grade is A
```

Syntax: Nested Switch Statement

Syntax

```
switch(choice 1) {
   case ‘A’:
   printf("This A is part of
        outer switch" );
   switch(choice 2) {
      case ‘A’:
      printf("This A is part
        of inner switch" );
      break;
      case ‘B’:
        /* case code */}
      break;
   case ‘B’: /* case code */
}
```

An example of the nested switch statement give the following result:

```
#include <stdio.h>
int main ()
{
/* local variable definition */
int a = 100;
int b = 200;
switch(a) {
case 100:
printf("This is part of outer switch\n");
switch(b) {
case 200:
printf("This is part of inner switch\n");
}
}
printf("Exact value of a is : %d\n", a );
printf("Exact value of b is : %d\n", b );
return 0;
}
```

OUTPUT

```
This is part of outer switch
This is part of inner switch
Exact value of a is : 100
Exact value of b is : 200
```

Switch Statement Exercise

1. Write a C program to input week number(1-7) and print day of week name using switch case.
2. Write a C program to input an alphabet and check whether it is vowel or consonant using switch case.
3. Write a C program to input month number and print total number of days in month using switch case.
4. Write a C program to create menu driven calculator that performs basic arithmetic operations (add, subtract, multiply and divide) using switch case and arithmetic operators.
5. Write a C program to verify a user ID and password. If both are verified print a welcome message to the user, otherwise print Incorrect Password. If the ID does not exist, the program should print Incorrect ID.