Computer Science
2210/22

This document covers every aspect of Pre-Release Material including detailed explanations, Pseudocodes along with their example running, efficiencies and expected questions.

PRE-RELEASE MATERIAL
MAY/JUNE
2022

0335-1400368 @gilanihaseeb Syed Haseeb Bari Gilani CS/IT - O/A Level

Pre-Release Material:

Your preparation for the examination should include attempting the following practical tasks by **writing and testing a program or programs**.

A program is needed to allow a Wildlife Park to sell tickets. A booking consists of one or more tickets for the same day(s) and can be made up to a week in advance. A booking can be made for a visit of one day or two consecutive days. A booking can have extra attractions included. A booking will be valid for the day(s) chosen only.

Ticket type	Cost for one day	Cost for two days
one adult	$20.00	$30.00
one child (an adult may bring up to two children)	$12.00	$18.00
one senior	$16.00	$24.00
family ticket (up to two adults or seniors, and three children)	$60.00	$90.00
groups of six people or more, price per person	$15.00	$22.50

Extra attraction	Cost per person
lion feeding	$2.50
penguin feeding	$2.00
evening barbecue (two-day tickets only)	$5.00

Write and test a program or programs for the Wildlife Park:

- Your program or programs must include appropriate prompts for the entry of data. Data must be validated on entry.
- All outputs, including error messages, need to be set out clearly and understandably.
- All variables, constants and other identifiers must have meaningful names.

You will need to complete these **three** tasks. Each task must be fully tested.

Task 1 – displaying the ticket options and the extra attractions available

Set up your program to:

- display the options, attractions and prices for one-day tickets
- display the options, attractions and prices for two-day tickets
- show the days available for booking; assume that there are tickets available for any valid day.

Task 2 – process a booking

Extend your program for **Task 1** to:

- input the tickets and extra attractions required, then calculate the total cost of the booking
- allocate a unique booking number
- display the booking details, including the total cost and the unique booking number
- repeat as required.

Task 3 – ensuring each booking is the best value

Check that the total for each booking gives the best value and offer an alternative if this is **not** the case. For example, buying two family tickets is better than a group ticket for a group of 10 that includes four adults and six children.

Main Idea of Pre-Release Material:

- This pre-release material contains a table with information of 5 different ticket types along with their prices and another table with information of 3 different extra attraction types along with their prices.
- It is based on a Wildlife Park which sells tickets.
- A booking can be made which consists of one or more tickets and it can be made either for one day or two consecutive days.
- The options of ticket types, attractions and prices are separately OUTPUT for one-day and two-day tickets both.
- The users can then choose from the displayed options and input their ticket type along with attractions required (users can buy as many tickets as they want).
- Every booking will be allocated a unique booking number.
- The total cost of the booking will be calculated (tickets cost + attractions cost) (prices used for calculation will be depending on either it is one-day booking or two-day booking) and it will be OUTPUT alongside the booking details and unique booking number.
- Lastly, the total cost for every booking will be checked to see if it is the cheapest possible total.
- If not, then an alternative ticketing approach would be offered that gives the cheapest possible total.

Explanation of Pre-Release Material:

A program is needed to allow a Wildlife Park to sell tickets. A booking consists of one or more tickets for the same day(s) and can be made up to a week in advance. A booking can be made for a visit of one day or two consecutive days.

It can be understood from this piece of text that:

- a single booking can have from one to as many tickets as user wants (no limit).
- it can be made on any day within a week (1-7 days before visit).
- it can be made for either 1-day visit or 2 consecutive days visit (all number of tickets in a single booking will be either 1-day or all number of tickets in a single booking will be 2-days without any exception).

A booking can have extra attractions included. A booking will be valid for the day(s) chosen only.

It can be understood from this piece of text that:

- users can choose extra attractions in their booking from the given 3 options (extra attractions are optional and charged per person).
- booking will be valid only for the day chosen (any 1 weekday) OR
- booking will be valid only for the days chosen (any 2 consecutive weekdays).

5 total ticket types

Separate tickets cost for 1-day and 2-days booking

Ticket type	Cost for one day	Cost for two days
one adult	$20.00	$30.00
one child (an adult may bring up to two children)	$12.00	$18.00
one senior	$16.00	$24.00
family ticket (up to two adults or seniors, and three children)	$60.00	$90.00
groups of six people or more, price per person	$15.00	$22.50

for every 1 adult, the child ticket limit variable will be incremented by 2 to make sure that the condition is satisfied. For example, 4 adults would mean that the statement:

child_ticket_limit ← child_ticket_limit + 2

will run 4 times and so:

child_ticket_limit = 0
child_ticket_limit ← 0 + 2

child_ticket_limit = 2
child_ticket_limit ← 2 + 2

child_ticket_limit = 4
child_ticket_limit ← 4 + 2

child_ticket_limit = 6
child_ticket_limit ← 6 + 2

child_ticket_limit = 8

therefore, 4 adults can bring up to 8 children and so the child_ticket_limit will then be compared with the number of children being input (using IF statement) to ensure that this condition is being satisfied.

WHILE loop will be used to check this condition and ensure that number of adults/seniors do not exceed 2 and number of children do not exceed 3.

WHILE loop will be used to check this condition and ensure that number of people in a group ticket are not less than 6.
Furthermore, the group ticket cost would be calculated by multiplying the price (1-day or 2-days) with number of people in a group ticket (price per individual person).

3 total extra attraction types

Separate attractions cost

Extra attraction	Cost per person
lion feeding	$2.50
penguin feeding	$2.00
evening barbecue (two-day tickets only)	$5.00

only users with 2-days booking will be exclusively asked and allowed for barbecue (condition checked using IF statement).

users with either 1-day booking or 2-days booking will both be asked and allowed for lion and penguin feeding.

Write and test a program or programs for the Wildlife Park:

- Your program or programs must include appropriate prompts for the entry of data. Data must be validated on entry.
- All outputs, including error messages, need to be set out clearly and understandably.
- All variables, constants and other identifiers must have meaningful names.

It can be understood from this piece of text that:

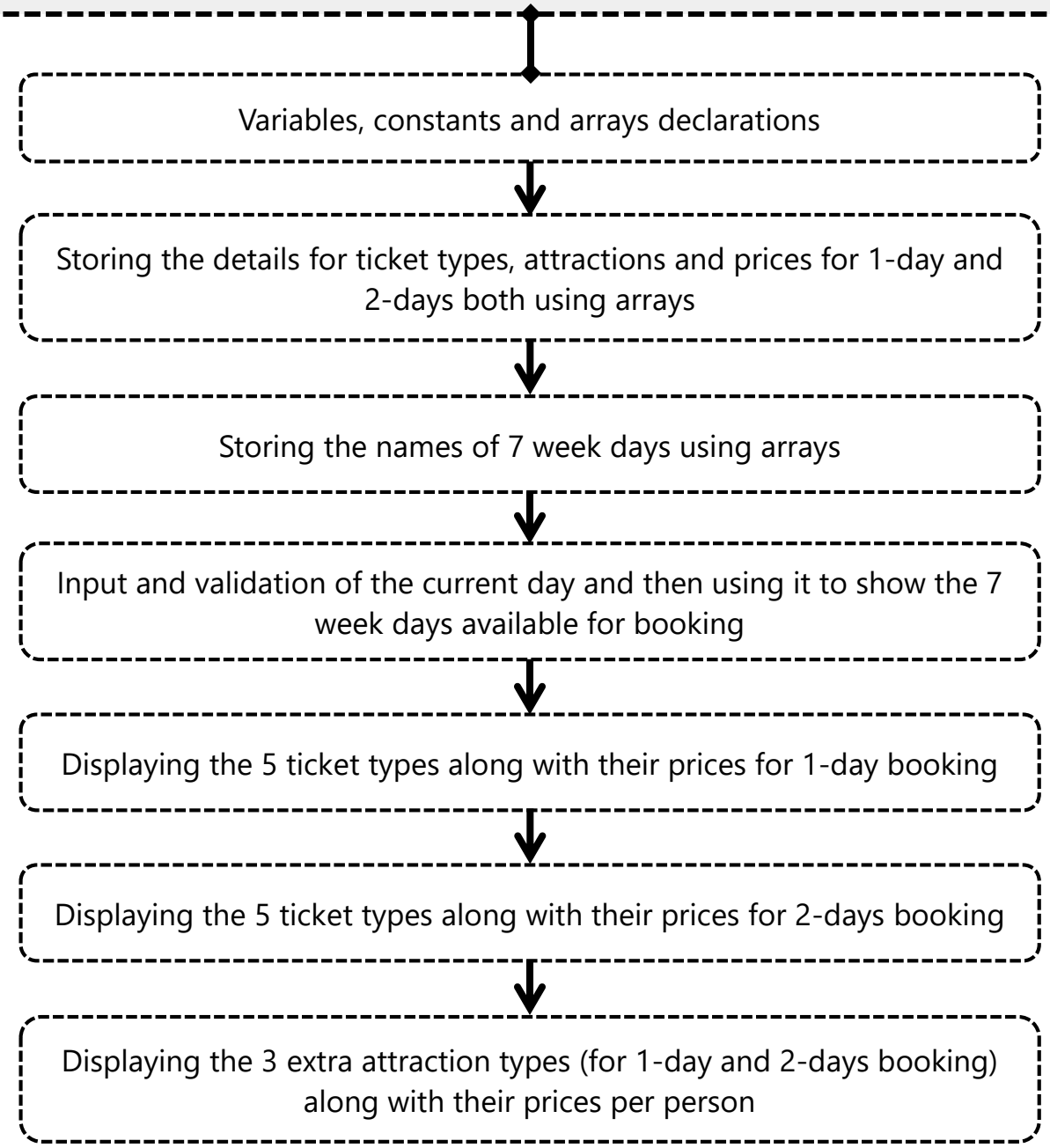
- the code must contain formal, suitable and clearly understandable messages/prompts that must be displayed when asking for input of data.
- the data must be validated through various checks and using selection statements ***(IF..THEN..END IF)*** and conditional loops ***(WHILE..DO..END WHILE)***
- if the input is wrong then the error message must be displayed and it should be formal, suitable and clearly understandable as well.
- all output of data must be displayed with proper messages/prompts describing what is the output showing or telling. They should be formal, suitable and clearly understandable as well.
- the program will use a number of arrays, variables and constants which must have clearly understandable and meaningful names that makes sense. (instead of using names such as $cost, the meaningful name must be used such as ***total_cost*** etc.

Concept and understanding of TASK 1:

Task 1 – displaying the ticket options and the extra attractions available

Set up your program to:

- display the options, attractions and prices for one-day tickets
- display the options, attractions and prices for two-day tickets
- show the days available for booking; assume that there are tickets available for any valid day.



Explanation of Algorithm of TASK 1:

In this task, we have to display options, attractions, prices (1-day and 2-days) and days available for booking along with declaring and initializing suitable data structures for **week days, ticket types**, **one day costs**, **two day costs**, **extra attraction types and extra attraction costs**.

- display the options, attractions and prices for one-day tickets

We will make use of **1D arrays to store the information** relating to 5 ticket types, 3 attractions and prices for **one-day tickets** (all of this information is given in the 2 tables given on Pre-release).

This is a demonstration of how all the data will be stored in the arrays according to index number:

Index	week_days
1	"Monday"
2	"Tuesday"
3	"Wednesday"
4	"Thursday"
5	"Friday"
6	"Saturday"
7	"Sunday"

ticket_type	one_day_cost	two_day_cost
"One adult"	20.00	30.00
"One child…"	12.00	18.00
"One senior"	16.00	24.00
"Family ticket…"	60.00	90.00
"Groups of…"	15.00	22.50

Index	extra_attraction_type	extra_attraction_cost
1	"Lion feeding"	2.50
2	"Penguin feeding"	2.00
3	"Evening barbecue…"	5.00

We will display the extra attractions and its prices collectively for both 1-day and 2-days booking in the end.

Therefore, for now, the display involves output of 5 ticket types and their prices for 1-day booking. It will be displayed/output using a ***FOR*** loop like this:

```
FOR count ← 1 TO 5
    PRINT "Ticket type: ", ticket_type[count], "Ticket cost for one-day booking: ",
          one_day_cost[count]
NEXT count
```

Running of example code:

When count will be 1:

```
PRINT "Ticket type: ", ticket_type[1]
PRINT "Ticket cost for one-day booking: ", one_day_cost[1]
```

Output will be:

```
Ticket type: One adult
Ticket cost for one-day booking: 20.00
```

When count will be 3:

```
PRINT "Ticket type: ", ticket_type[3]
PRINT "Ticket cost for one-day booking: ", one_day_cost[3]
```

Output will be:

```
Ticket type: One senior
Ticket cost for one-day booking: 16.00
```

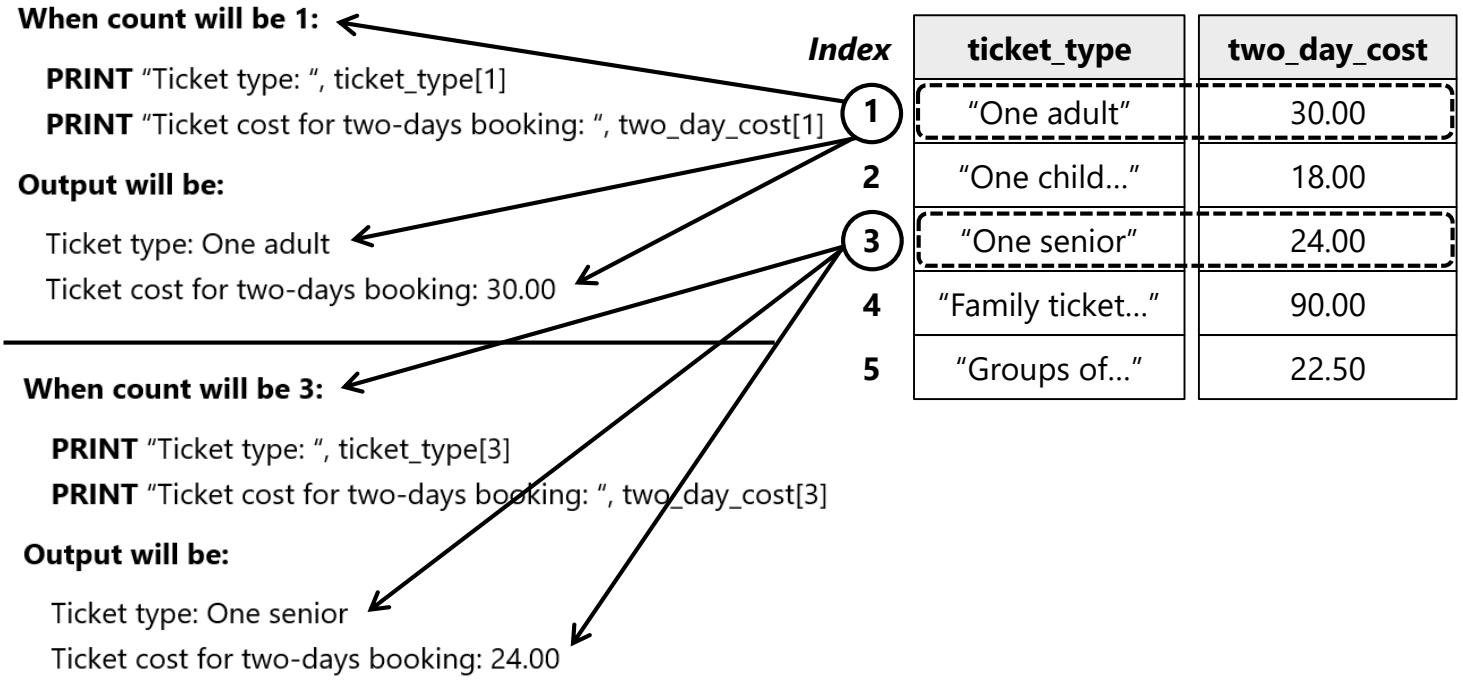
Index	ticket_type	one_day_cost
1	"One adult"	20.00
2	"One child..."	12.00
3	"One senior"	16.00
4	"Family ticket..."	60.00
5	"Groups of..."	15.00

- display the options, attractions and prices for two-day tickets

Secondly we will output the 5 ticket types and their prices for 2-days booking using a similar ***FOR*** loop like this:

```
FOR count ← 1 TO 5
    PRINT "Ticket type: ", ticket_type[count], "Ticket cost for two-days booking: ",
          two_day_cost[count]
NEXT count
```

Running of example code:



In this way, the ticket types and their prices for **1-day booking** and **2-days booking** will be taken from their locations in the arrays (after being searched according to index value) and **PRINTED/OUTPUT** separately.

Lastly, we will display the extra attractions and its prices collectively for both 1-day and 2-days booking. So the display would involve output of 3 attraction types and their prices for 1-day and 2-days booking. It will be displayed/output in the same manner using a ***FOR*** loop like shown above.

- show the days available for booking; assume that there are tickets available for any valid day

We will make use of **1D arrays to store the information** relating to **7 week days**. The user will then be asked to input number of current day (e.g. 1 for Monday, 2 for Tuesday, 7 for Sunday and so on.). A **WHILE** loop will be used for validation and to ensure that any number from 1 to 7 is being entered. Input of any other number e.g. less than 1 or greater than 7 will output an error message like this:

```
INPUT "What day is it today? Input 1 for Monday, 2 for Tuesday, 3 for Wednesday, 4 for
      Thursday, 5 for Friday, 6 for Saturday and 7 for Sunday.", today
WHILE today < 1 OR today > 7
   INPUT "Wrong input. Kindly enter what day is it today? Input 1 for Monday, 2 for Tuesday, 3 for
         Wednesday, 4 for Thursday, 5 for Friday, 6 for Saturday and 7 for Sunday.", today
END WHILE
```

Then the entered number would be used as an index value and so the week day stored at that location will be searched and stored in a variable named ***current_day*:**

current_day ← week_days[today]

So if the user has input 4 in the variable ***today*** then the data stored at that location will be searched and stored in ***current_day***:

week_days[4] ← "Thursday"
current_day ← week_days[4]
current_day ← "Thursday"

Therefore, the following piece of code will show the 7 days available for booking:

PRINT "The upcoming 7 days are available for booking before the next: ", current_day

TASK 1 – Pseudocode:

BEGIN

```
DECLARE week_days [1:7] AS STRING
DECLARE ticket_type [1:5], extra_attraction_type [1:3] AS STRING
DECLARE one_day_cost [1:5], two_day_cost [1:5], extra_attraction_cost [1:3] AS FLOAT
DECLARE today ← 0 AS INTEGER
DECLARE current_day ← "" AS STRING

week_days[1] ← "Monday"
week_days[2] ← "Tuesday"
week_days[3] ← "Wednesday"
week_days[4] ← "Thursday"
week_days[5] ← "Friday"
week_days[6] ← "Saturday"
week_days[7] ← "Sunday"

ticket_type[1] ← "One adult"
ticket_type[2] ← "One child (an adult may bring up to two children)"
ticket_type[3] ← "One senior"
ticket_type[4] ← "Family ticket (up to two adults or seniors, and three children)"
ticket_type[5] ← "Groups of six people or more (price charged per person)"

one_day_cost[1] ← 20.00
one_day_cost[2] ← 12.00
one_day_cost[3] ← 16.00
one_day_cost[4] ← 60.00
one_day_cost[5] ← 15.00

two_day_cost[1] ← 30.00
two_day_cost[2] ← 18.00
two_day_cost[3] ← 24.00
two_day_cost[4] ← 90.00
two_day_cost[5] ← 22.50

extra_attraction_type[1] ← "Lion feeding"
extra_attraction_type[2] ← "Penguin feeding"
extra_attraction_type[3] ← "Evening barbecue (for two-day tickets only)"

extra_attraction_cost[1] ← 2.50
extra_attraction_cost[2] ← 2.00
extra_attraction_cost[3] ← 5.00
```

```
INPUT "What day is it today? Input 1 for Monday, 2 for Tuesday, 3 for Wednesday, 4 for
      Thursday, 5 for Friday, 6 for Saturday and 7 for Sunday.", today
WHILE today < 1 OR today > 7
   INPUT "Wrong input. Kindly enter what day is it today? Input 1 for Monday, 2 for Tuesday, 3 for
         Wednesday, 4 for Thursday, 5 for Friday, 6 for Saturday and 7 for Sunday.", today
END WHILE

current_day ← week_days[today]

PRINT "Welcome to the Wildlife Park!"
PRINT "The following are 5 ticket types (options) along with their prices for one-day booking:"
FOR count ← 1 TO 5
   PRINT "Ticket type: ", ticket_type[count], "Ticket cost for one-day booking: ", one_day_cost[count]
NEXT count

PRINT "The following are 5 ticket types (options) along with their prices for two-days booking:"
FOR count ← 1 TO 5
   PRINT "Ticket type: ", ticket_type[count], "Ticket cost for two-days booking: ", two_day_cost[count]
NEXT count

PRINT "The following are 3 extra attraction types (options) along with their prices for one-day and
      two-days booking:"
FOR count ← 1 TO 3
   PRINT "Extra attractions: ", extra_attraction_type[count], "Cost per person: ",
         extra_attraction_cost[count]
NEXT count

PRINT "The upcoming 7 days are available for booking before the next: ", current_day

END
```

TASK 1 – Efficiency:

- Use of ***ARRAYS*** to store ticket types, extra attractions and week days.
- Use of different ***ARRAYS*** to store prices for one-day and two-day bookings separately.
- Initialization of all ***ARRAYS*** with **pre-defined values**.
- Use of ***FOR*** loops to output details of ticket types, attraction types and their prices.
- Use of ***WHILE*** loop to validate user input and output appropriate error messages when validation fails.

TASK 1 – Explanation of Pseudocode:

```
DECLARE week_days [1:7] AS STRING
DECLARE ticket_type [1:5], extra_attraction_type [1:3] AS STRING
DECLARE one_day_cost [1:5], two_day_cost [1:5], extra_attraction_cost [1:3] AS FLOAT
DECLARE today ← 0 AS INTEGER
DECLARE current_day ← "" AS STRING
```

Declaration of variables, constants and arrays.

```
week_days[1] ← "Monday"
week_days[2] ← "Tuesday"
week_days[3] ← "Wednesday"
week_days[4] ← "Thursday"
week_days[5] ← "Friday"
week_days[6] ← "Saturday"
week_days[7] ← "Sunday"

ticket_type[1] ← "One adult"
ticket_type[2] ← "One child (an adult may bring up to two children)"
ticket_type[3] ← "One senior"
ticket_type[4] ← "Family ticket (up to two adults or seniors, and three children)"
ticket_type[5] ← "Groups of six people or more (price charged per person)"

one_day_cost[1] ← 20.00
one_day_cost[2] ← 12.00
one_day_cost[3] ← 16.00
one_day_cost[4] ← 60.00
one_day_cost[5] ← 15.00

two_day_cost[1] ← 30.00
two_day_cost[2] ← 18.00
two_day_cost[3] ← 24.00
two_day_cost[4] ← 90.00
two_day_cost[5] ← 22.50

extra_attraction_type[1] ← "Lion feeding"
extra_attraction_type[2] ← "Penguin feeding"
extra_attraction_type[3] ← "Evening barbecue (for two-day tickets only)"

extra_attraction_cost[1] ← 2.50
extra_attraction_cost[2] ← 2.00
extra_attraction_cost[3] ← 5.00
```

Assigning pre-defined values to 6 different arrays.

```
INPUT "What day is it today? Input 1 for Monday, 2 for Tuesday, 3 for Wednesday, 4 for
      Thursday, 5 for Friday, 6 for Saturday and 7 for Sunday.", today
WHILE today < 1 OR today > 7
   INPUT "Wrong input. Kindly enter what day is it today? Input 1 for Monday, 2 for Tuesday, 3 for
         Wednesday, 4 for Thursday, 5 for Friday, 6 for Saturday and 7 for Sunday.", today
END WHILE
```

Input and validation of current day

```
current_day ← week_days[today]
```

Calculation of days available for booking

```
PRINT "Welcome to the Wildlife Park!"
PRINT "The following are 5 ticket types (options) along with their prices for one-day booking:"
FOR count ← 1 TO 5
   PRINT "Ticket type: ", ticket_type[count], "Ticket cost for one-day booking: ", one_day_cost[count]
NEXT count
```

Use of FOR loop to output details for one-day booking

```
PRINT "The following are 5 ticket types (options) along with their prices for two-days booking:"
FOR count ← 1 TO 5
   PRINT "Ticket type: ", ticket_type[count], "Ticket cost for two-days booking: ", two_day_cost[count]
NEXT count
```

Use of FOR loop to output details for two-day booking

```
PRINT "The following are 3 extra attraction types (options) along with their prices for one-day and
      two-days booking:"
FOR count ← 1 TO 3
   PRINT "Extra attractions: ", extra_attraction_type[count], "Cost per person: ",
         extra_attraction_cost[count]
NEXT count
```

Use of FOR loop to output details for extra attractions

```
PRINT "The upcoming 7 days are available for booking before the next: ", current_day
```

TASK 1 – Expected Questions:

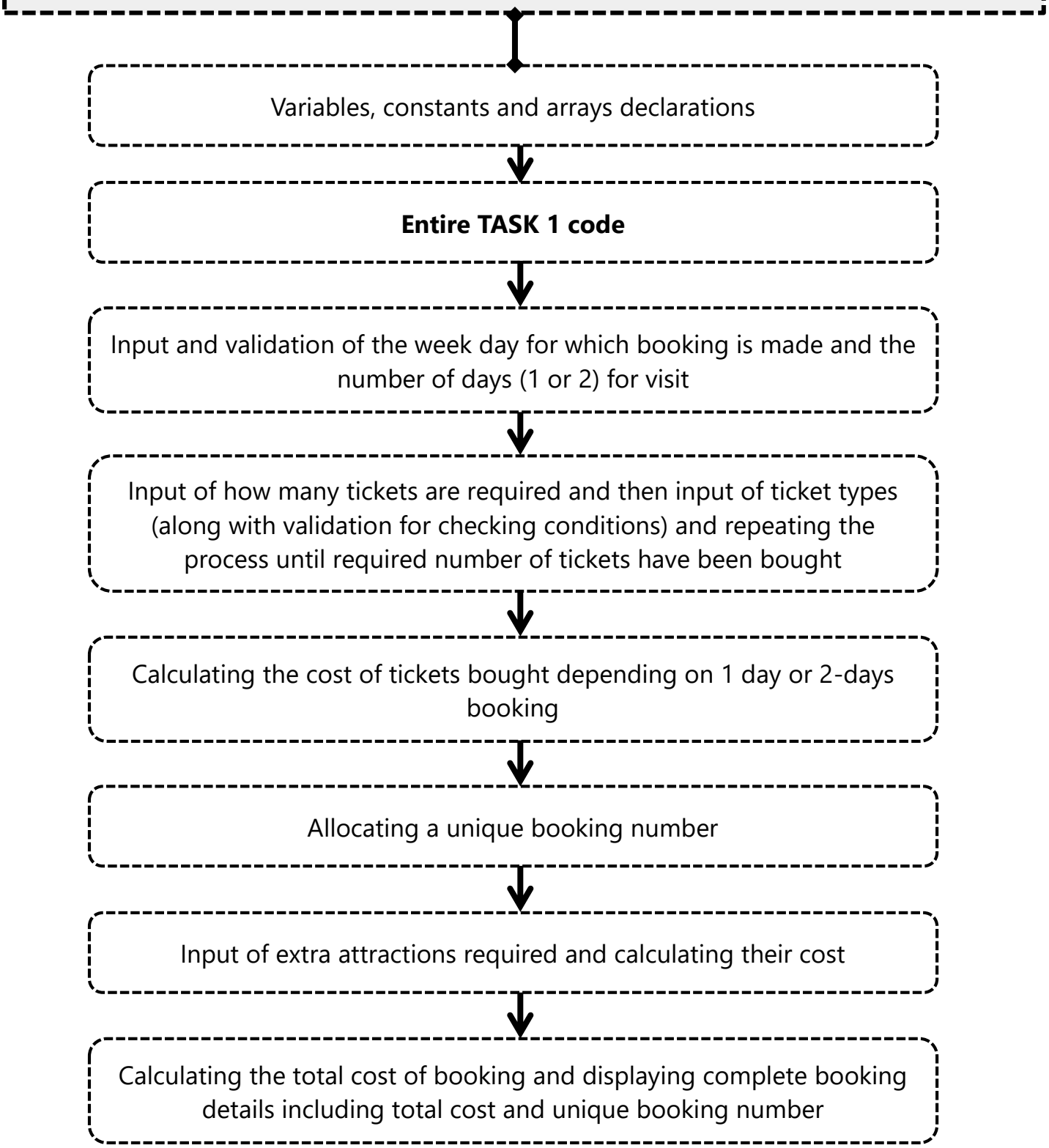
1. State three arrays you used for Task 1. State the data type and purpose of the arrays.
2. Describe the data structures you have used in Task 1 to store the data for the park. Include the name(s), data type, sample data and usage for each structure.
3. Write an algorithm for Task 1, using either Pseudocode, programming statements or a flowchart.
4. Write an algorithm for Task 1, using either Pseudocode, programming statements or a flowchart. Assume that the data structures for storing data have already been initialized with predefined values.
5. Write an algorithm to complete Task 1 without including any appropriate prompts, using either Pseudocode, programming statements or a flowchart.
6. Explain how your program completes/performs Task 1. Any programming statements used in your answer must be fully explained.
7. Explain how you calculated the days available for booking (part of Task 1)? You can include Pseudocode or programming statements as part of your explanation.
8. Comment on the efficiency of your code for Task 1.

Concept and understanding of TASK 2:

Task 2 – process a booking

Extend your program for **Task 1** to:

- input the tickets and extra attractions required, then calculate the total cost of the booking
- allocate a unique booking number
- display the booking details, including the total cost and the unique booking number
- repeat as required.



Explanation of Algorithm of TASK 2:

In this task, we have to input the tickets required and extra attractions. Then calculate the total cost booking and allocate a unique booking number. In the end, we will simply output all the booking details.

- input the tickets and extra attractions required, then calculate the total cost of the booking

Firstly, we will input the week day for which user wants to make the booking. Then we will input the number of days for which booking is required. It would be either:

- 1-day booking OR
- 2-days booking

A ***WHILE*** loop will be used for validation and to ensure that only **"1"** or **"2"** are being entered by the user. Input of any other number of days at this specific stage will output an error message like this:

```
INPUT "Kindly enter the number of days for which the booking is required. Input 1 for one day
      booking or 2 for two consecutive days booking", no_of_days
WHILE no_of_days < 1 OR no_of_days > 2
  INPUT "Wrong input. Kindly enter again.", no_of_days
END WHILE
```

- repeat as required

The user will then be asked to input how many tickets they want to buy. Then accordingly a ***REPEAT*** loop will be used to ensure that the whole process of purchasing a ticket keeps repeating itself until the user has bought the required number of tickets:

```
REPEAT
    INPUT "How many tickets would you like to buy?", no_of_tickets
        ENTIRE TICKET SELECTION CODE
    ticket_count ← ticket_count + 1
UNTIL ticket_count = no_of_tickets
```

A ticket counter is incremented every time the ***REPEAT*** loop (ticket selection code) is run to record the number of tickets purchased.

The entire ticket selection code will run ***UNTIL*** the ticket counter has become equal to the number of tickets the user wanted to buy (as input by them).

The user will then be asked to input any 1 ticket type out of 5 (according to their choice). A ***WHILE*** loop will be used for validation and to ensure that only **"1", "2", "3", "4" or "5"** are being entered by the user. Input of any other number for choosing ticket at this specific stage will output an error message like this:

```
INPUT "What type of the ticket would you like to buy? Input 1 for Adult Ticket, 2 for Child Ticket,
      3 for Senior Ticket, 4 for Family Ticket or 5 for Group Ticket.", selected_ticket_type
WHILE selected_ticket_type < 1 OR selected_ticket_type > 5
  INPUT "Wrong input. Kindly enter again.", selected_ticket_type
END WHILE
```

The following 5 explanations cover the algorithm for all 5 possible inputs to the variable **selected_ticket_type**, calculation of the selected ticket prices and the logic behind each statement used in the code:

(i) Adult Ticket:

IF statement is used to check that **selected_ticket_type** = 1. If this is TRUE, then:

```
IF selected_ticket_type = 1 THEN
    adult_ticket_count ← adult_ticket_count + 1
    child_ticket_limit ← child_ticket_limit + 2
```

The ticket counter for adult ticket type would be incremented with purchase of every 1 adult ticket. There will also be a child ticket limit which would be incremented by 2 simultaneously every time an adult ticket is purchased to ensure that the condition given in Pre-release is being satisfied:

condition: an adult may bring up to two children

Example Working:

For every 1 adult, the child ticket limit variable will be incremented by 2 to make sure that the condition is satisfied. For example, 3 adults would mean that the statement:

child_ticket_limit ← child_ticket_limit + 2

will run 3 times and so:

child_ticket_limit = 0
child_ticket_limit ← 0 + 2

child_ticket_limit = 2
child_ticket_limit ← 2 + 2

child_ticket_limit = 4
child_ticket_limit ← 4 + 2

child_ticket_limit = 6

Therefore, 3 adults can bring up to 6 children and so the **child_ticket_limit** will then be compared with the number of children being input (using ***IF*** statement) to ensure that this condition is being satisfied.

Another ***IF*** statement is used to check that **no_of_days** = 1. If this is TRUE, then the cost for one-day booking will be used to calculate the price of ticket. If this is FALSE, then the cost for two-days booking will be used to calculate the price of ticket as given below:

```
IF no_of_days = 1 THEN
    adult_cost ← adult_ticket_count * one_day_cost[1]
ELSE
    adult_cost ← adult_ticket_count * two_day_cost[1]
END IF
```

The **adult_ticket_count** would store the number of adult tickets purchased and it will also be used to calculate the cost of adult tickets.

For example, if 7 adult tickets were bought then:

- adult_ticket_count ← 7
- child_ticket_limit ← 14

IF no_of_days = 1 **THEN**

- adult_cost ← 7 * 20.00
- adult_cost ← 140

ELSE (no_of_days = 2)

- adult_cost ← 7 * 30.00
- adult_cost ← 210

(ii) Child Ticket:

IF statement is used to check that **selected_ticket_type** = 2. If this is TRUE, then:

```
IF child_ticket_limit = 0 THEN
    PRINT "The current number of adults cannot bring more children."
ELSE
    child_ticket_limit ← child_ticket_limit - 1
    child_ticket_count ← child_ticket_count + 1
END IF
```

The **child_ticket_limit** would be checked for if it is equal to 0. If the child ticket limit is 0 then then the user will not be allowed to purchase the ticket as the limit has been reached.

If this is FALSE, then the child ticket counter would be incremented and the child ticket limit would be decremented as another child ticket has been purchased therefore reducing the limit.

Another ***IF*** statement is used to check that **no_of_days** = 1. If this is TRUE, then the cost for one-day booking will be used to calculate the price of ticket. If this is FALSE, then the cost for two-days booking will be used to calculate the price of ticket.

The **child_ticket_count** would store the number of child tickets purchased and it will also be used to calculate the cost of child tickets.

(iii) Senior Ticket:

IF statement is used to check that **selected_ticket_type** = 3. If this is TRUE, then **senior_ticket_count** would be incremented.

Another ***IF*** statement is used to check that **no_of_days** = 1. If this is TRUE, then the cost for one-day booking will be used to calculate the price of ticket. If this is FALSE, then the cost for two-days booking will be used to calculate the price of ticket.

The **senior_ticket_count** would store the number of child tickets purchased and it will also be used to calculate the cost of senior tickets.

(iv) Family Ticket:

IF statement is used to check that **selected_ticket_type** = 4. If this is TRUE, then **family_ticket_count** would be incremented.

The following variables would be used for storing the number of adults, seniors and children:

- **no_of_adults**
- **no_of_seniors**
- **no_of_children**

The user would be asked to input the number of adults (max 2). A ***WHILE*** loop will be used for validation and to ensure that minimum: 0 and maximum: 2 adults are only being entered.

IF statement would be used to check that if **no_of_adults** = 0 then the user would be asked to input the number of seniors (max 2). A ***WHILE*** loop will be used for validation and to ensure that minimum: 0 and maximum: 2 seniors are only being entered.

Another ***IF*** statement would be used to check that if **no_of_adults** = 1 then the user would be asked to input the number of seniors (max 1). A ***WHILE*** loop will be used for validation and to ensure that minimum: 0 and maximum: 1 senior are only being entered.

All of this is done to ensure that the following condition is being satisfied:

condition: up to two adults or seniors, and three children

So the possible combinations of adults and seniors are:

- 2 adults
- 2 seniors
- 1 adult and 1 senior

The user would then be asked to input the number of children (max 3). A ***WHILE*** loop will be used for validation and to ensure that minimum: 0 and maximum: 3 children are only being entered.

Another ***IF*** statement is used to check that **no_of_days** = 1. If this is TRUE, then the cost for one-day booking will be used to calculate the price of ticket. If this is FALSE, then the cost for two-days booking will be used to calculate the price of ticket.

The **family_ticket_count** would store the number of family tickets purchased and it will also be used to calculate the cost of family tickets.

(v) Group Ticket:

IF statement is used to check that **selected_ticket_type** = 5. If this is TRUE, then **group_no_of_people** would be input and validated using ***WHILE*** loop to ensure that it is greater than or equal to 6.

The total people in a group will be totaled and updated every time a group ticket is purchased to easily calculate the price per person for all the people in a group:

```
group_total_people ← group_total_people + group_no_of_people
```

The following variables would be used for storing the number of adults, seniors and children:

- **group_no_of_adults**
- **group_no_of_seniors**
- **group_no_of_children**

(i) The user would be asked to input the number of adults. A ***WHILE*** loop will be used for validation and to ensure that the number of adults do not exceed the number of total people in the group:

```
INPUT "Kindly enter the number of adults in the group.", group_no_of_adults
WHILE group_no_of_adults > group_no_of_people
    INPUT "Wrong input. Kindly enter again.", group_no_of_adults
END WHILE
```

The variable **group_no_of_people** would be subtracted from the number of adults entered and the child ticket limit would be updated/increased according to the number of adults entered:

```
child_ticket_limit ← child_ticket_limit + (group_no_of_adults * 2)
group_no_of_people ← group_no_of_people – group_no_of_adults
group_total_adults ← group_total_adults + group_no_of_adults
```

(ii) The user would then be asked to input the number of children. A ***WHILE*** loop will be used for validation and to ensure that the number of children do not exceed the number of total people in the group as well as the child ticket limit (this is done to ensure that the **condition: an adult may bring up to two children** is satisfied):

```
INPUT "Kindly enter the number of children in the group.", group_no_of_children
WHILE group_no_of_children > child_ticket_limit AND group_no_of_children >
        group_no_of_people
    INPUT "Wrong input. Kindly enter again.", group_no_of_children
END WHILE
```

The variable **group_no_of_people** would be subtracted from the number of children entered and the child ticket limit would be reduced according to the number of children entered:

```
child_ticket_limit ← child_ticket_limit – group_no_of_children
group_no_of_people ← group_no_of_people – group_no_of_children
group_total_children ← group_total_children + group_no_of_children
```

(iii) The user would finally be asked to input the number of seniors. Similarly, a ***WHILE*** loop will be used for validation and to ensure that the number of seniors do not exceed the number of total people in the group.

In the same manner, the variable **group_no_of_people** would be subtracted from the number of seniors entered.

This whole process will be repeated using ***REPEAT*** loop ***UNTIL*** the **group_no_of_people** = 0 as this variable was continuously decreasing according to the number of adults, seniors and children being entered.

For example, if **group_no_of_people** = 10:

- and group_no_of_adults = 4 so group_no_of_people = 10 – 4
- then group_no_of_people ← 6
- and group_no_of_seniors = 6 so group_no_of_people = 6 – 6
- then group_no_of_people ← 0

Therefore, after input of 4 adults and 6 seniors (total 10 people), the group_no_of_people would become 0 and hence the loop will end as chosen number of people have been entered.

IF statement is used to check that **no_of_days** = 1. If this is TRUE, then the cost for one-day booking will be used to calculate the price of ticket. If this is FALSE, then the cost for two-days booking will be used to calculate the price of ticket.

The **group_total_people** would store the number of persons in a group and it will also be used to calculate the price per person of everyone in the group.

The whole process of purchasing a ticket keeps repeating itself until the user has bought the required number of tickets.

Then the user will be asked for input of extra attractions required and their cost would be calculated simultaneously. The first 2 attractions would be simply input and their cost calculated:

```
INPUT "Kindly enter the number of persons who want to feed a lion", lion_feeding_persons
      lion_feeding_cost ← lion_feeding_persons * extra_attraction_cost[1]

INPUT "Kindly enter the number of persons who want to feed a penguin",
      penguin_feeding_persons
      penguin_feeding_cost ← penguin_feeding_persons * extra_attraction_cost[2]
```

The 3rd attraction would be offered based upon whether it is 1-day or 2-days booking. To ensure if this is a 2-days booking, ***IF*** statement would be used and then accordingly the last attraction would be input and its cost calculated:

```
IF no_of_days = 2 THEN
    INPUT "Kindly enter the number of persons who want to do an evening barbecue",
          barbecue_persons
          barbecue_cost ← barbecue_persons * extra_attraction_cost[3]
END IF
```

The variables used for extra attractions cost:

- **lion_feeding_cost**
- **penguin_feeding_cost**
- **barbecue_cost**

And the variables used for ticket types cost:

- **adult_cost**
- **child_cost**
- **senior_cost**
- **family_cost**
- **group_cost**

Will all be added together and then totaled in the following way:

```
total_attractions_cost ← lion_feeding_cost + penguin_feeding_cost + barbecue_cost
total_tickets_cost ← adult_cost + child_cost + senior_cost + family_cost + group_cost

total_booking_cost ← total_attractions_cost + total_tickets_cost
```

This would complete the calculation of the total cost for booking.

- allocate a unique booking number

The method used for calculating and then allocating a unique booking number is kept very simple. It is similar to a counter being incremented (+1) with every single booking to ensure its unique as well as easy:

```
unique_booking_no ← unique_booking_no + 1
```

- display the booking details, including the total cost and the unique booking number

The following booking details will be **displayed/OUTPUT**:

- The unique booking number
- The week day for which booking is made
- The number of day(s) for which booking is made
- The number of ticket(s) bought
- The number of adult, child, senior and family ticket(s) bought
- The number of people who are a part of group ticket(s)
- The number of people who selected extra attractions like lion feeding, penguin feeding and evening barbecue
- The separate total cost for extra attractions
- The separate total cost for tickets bought
- The grand total cost for complete booking (extra attractions + tickets cost)

TASK 2 – Pseudocode:

BEGIN

[ALL IDENTIFIERS OF TASK 1]

```
DECLARE booking_day ← 0, no_of_days ← 0, no_of_tickets ← 0, ticket_count ← 0 AS INTEGER
DECLARE unique_booking_no ← 0, selected_ticket_type ← 0 AS INTEGER
DECLARE adult_ticket_count ← 0, child_ticket_count ← 0, senior_ticket_count ← 0
        family_ticket_count ← 0 AS INTEGER
DECLARE child_ticket_limit ← 0, group_no_of_people ← 0 AS INTEGER
DECLARE no_of_adults ← 0, no_of_seniors ← 0, no_of_children ← 0 AS INTEGER
DECLARE group_no_of_adults ← 0, group_no_of_seniors ← 0, group_no_of_children ← 0 AS INTEGER
DECLARE group_total_adults ← 0, group_total_children ← 0, group_total_seniors ← 0,
        group_total_people ← 0 AS INTEGER
DECLARE adult_cost ← 0.0, child_cost ← 0.0, senior_cost ← 0.0, family_cost ← 0.0 group_cost ← 0.0
        AS FLOAT
DECLARE lion_feeding_persons ← 0, penguin_feeding_persons ← 0, barbecue_persons ← 0 AS INTEGER
DECLARE lion_feeding_cost ← 0.0, penguin_feeding_cost ← 0.0, barbecue_cost ← 0.0 AS FLOAT
DECLARE total_attractions_cost ← 0.0, total_tickets_cost ← 0.0, total_booking_cost ← 0.0 AS FLOAT
```

[ENTIRE PSEUDOCODE OF TASK 1]

```
INPUT "Kindly enter the week day for which you want to make a booking. Input 1 for Monday, 2
      for Tuesday, 3 for Wednesday, 4 for Thursday, 5 for Friday, 6 for Saturday and 7 for Sunday."
      booking_day
WHILE booking_day < 1 OR booking_day > 7
   INPUT "Wrong input. Kindly enter again.", booking_day
END WHILE

INPUT "Kindly enter the number of days for which the booking is required. Input 1 for one day booking
      or 2 for two consecutive days booking", no_of_days
WHILE no_of_days < 1 OR no_of_days > 2
   INPUT "Wrong input. Kindly enter again.", no_of_days
END WHILE

REPEAT

   INPUT "How many tickets would you like to buy?", no_of_tickets

   INPUT "What type of the ticket would you like to buy? Input 1 for Adult Ticket, 2 for Child Ticket,
         3 for Senior Ticket, 4 for Family Ticket or 5 for Group Ticket.", selected_ticket_type
   WHILE selected_ticket_type < 1 OR selected_ticket_type > 5
      INPUT "Wrong input. Kindly enter again.", selected_ticket_type
   END WHILE
```

```
IF selected_ticket_type = 1 THEN
    adult_ticket_count ← adult_ticket_count + 1
    child_ticket_limit ← child_ticket_limit + 2

    IF no_of_days = 1 THEN
        adult_cost ← adult_ticket_count * one_day_cost[1]
    ELSE
        adult_cost ← adult_ticket_count * two_day_cost[1]
    END IF
END IF

IF selected_ticket_type = 2 THEN

    IF child_ticket_limit = 0 THEN
        PRINT "The current number of adults cannot bring more children."
    ELSE
        child_ticket_limit ← child_ticket_limit – 1
        child_ticket_count ← child_ticket_count + 1
    END IF

    IF no_of_days = 1 THEN
        child_cost ← child_ticket_count * one_day_cost[2]
    ELSE
        child_cost ← child_ticket_count * two_day_cost[2]
    END IF
END IF

IF selected_ticket_type = 3 THEN
    senior_ticket_count ← senior_ticket_count + 1

    IF no_of_days = 1 THEN
        senior_cost ← senior_ticket_count * one_day_cost[3]
    ELSE
        senior_cost ← senior_ticket_count * two_day_cost[3]
    END IF
END IF

IF selected_ticket_type = 4 THEN
    family_ticket_count ← family_ticket_count + 1

    INPUT "Kindly enter the number of adults. Up to two adults are allowed.", no_of_adults
    WHILE no_of_adults < 0 OR no_of_adults > 2
        INPUT "Wrong input. Kindly enter again.", no_of_adults
    END WHILE
```

```
    IF no_of_adults = 0 THEN
        INPUT "Kindly enter the number of seniors. Up to two seniors are allowed.",
              no_of_seniors
        WHILE no_of_seniors < 0 OR no_of_seniors > 2
            INPUT "Wrong input. Kindly enter again.", no_of_seniors
        END WHILE
    END IF

    IF no_of_adults = 1 THEN
        INPUT "Kindly enter the number of seniors. Only one senior is allowed.", no_of_seniors
        WHILE no_of_seniors < 0 OR no_of_seniors > 1
            INPUT "Wrong input. Kindly enter again.", no_of_seniors
        END WHILE
    END IF

    INPUT "Kindly enter the number of children. Up to three children are allowed.", no_of_children
    WHILE no_of_children < 0 OR no_of_children > 3
        INPUT "Wrong input. Kindly enter again.", no_of_children
    END WHILE

    IF no_of_days = 1 THEN
        family_cost ← family_ticket_count * one_day_cost[4]
    ELSE
        family_cost ← family_ticket_count * two_day_cost[4]
    END IF
END IF

IF selected_ticket_type = 5 THEN
    INPUT "Kindly enter the total number of people in the group (six or more people).",
          group_no_of_people
    WHILE  group_no_of_people < 6
        INPUT "Wrong input. Kindly enter again.", group_no_of_people
    END WHILE
    group_total_people ← group_total_people + group_no_of_people

    REPEAT
        INPUT "Kindly enter the number of adults in the group.", group_no_of_adults
        WHILE group_no_of_adults > group_no_of_people
            INPUT "Wrong input. Kindly enter again.", group_no_of_adults
        END WHILE
```

```
            child_ticket_limit ← child_ticket_limit + (group_no_of_adults * 2)
            group_no_of_people ← group_no_of_people – group_no_of_adults
            group_total_adults ← group_total_adults + group_no_of_adults

            INPUT "Kindly enter the number of children in the group.", group_no_of_children
            WHILE group_no_of_children > child_ticket_limit AND group_no_of_children >
                    group_no_of_people
               INPUT "Wrong input. Kindly enter again.", group_no_of_children
            END WHILE
            child_ticket_limit ← child_ticket_limit – group_no_of_children
            group_no_of_people ← group_no_of_people – group_no_of_children
            group_total_children ← group_total_children + group_no_of_children

            INPUT "Kindly enter the number of seniors in the group.", group_no_of_seniors
            WHILE group_no_of_seniors > group_no_of_people
               INPUT "Wrong input. Kindly enter again.", group_no_of_seniors
            END WHILE
            group_no_of_people ← group_no_of_people – group_no_of_seniors
            group_total_seniors ← group_total_seniors + group_no_of_seniors
        UNTIL group_no_of_people = 0

        IF no_of_days = 1 THEN
           group_cost ← group_total_people * one_day_cost[5]
        ELSE
           group_cost ← group_total_people * two_day_cost[5]
        END IF
    END IF

ticket_count ← ticket_count + 1

UNTIL ticket_count = no_of_tickets

PRINT "Kindly enter the following details if you want to have extra attractions included:"
INPUT "Kindly enter the number of persons who want to feed a lion: ", lion_feeding_persons
      lion_feeding_cost ← lion_feeding_persons * extra_attraction_cost[1]
INPUT "Kindly enter the number of persons who want to feed a penguin: ", penguin_feeding_persons
      penguin_feeding_cost ← penguin_feeding_persons * extra_attraction_cost[2]
IF no_of_days = 2 THEN
   INPUT "Kindly enter the number of persons who want to do an evening barbecue", barbecue_persons
         barbecue_cost ← barbecue_persons * extra_attraction_cost[3]
END IF
```

```
unique_booking_no ← unique_booking_no + 1

total_attractions_cost ← lion_feeding_cost + penguin_feeding_cost + barbecue_cost
total_tickets_cost ← adult_cost + child_cost + senior_cost + family_cost + group_cost

total_booking_cost ← total_attractions_cost + total_tickets_cost

PRINT "The following are your booking details:"

PRINT "The unique booking number is: ", unique_booking_no
PRINT "The booking is made for the following week day: ", booking_day
PRINT "The booking is only valid for following number of day(s): ", no_of_days
PRINT "The following number of ticket(s) were bought: ", no_of_tickets
PRINT "The following number of adult ticket(s) were bought: ", adult_ticket_count
PRINT "The following number of child ticket(s) were bought: ", child_ticket_count
PRINT "The following number of senior ticket(s) were bought: ", senior_ticket_count
PRINT "The following number of family ticket(s) were bought: ", family_ticket_count
PRINT "The following number of people are part of group ticket(s): ", group_total_people
PRINT "The following number of person(s) want to feed a lion: ", lion_feeding_persons
PRINT "The following number of person(s) want to feed a penguin: ", penguin_feeding_persons
PRINT "The following number of person(s) want to do an evening barbecue: ", barbecue_persons
PRINT "The following is the total cost for extra attractions: ", total_attractions_cost
PRINT "The following is the total cost for tickets bought: ", total_tickets_cost
PRINT "The following is the grand total cost for complete booking: ", total_booking_cost

END
```

TASK 2 – Efficiency:

- Use of ***WHILE*** loops to validate all user inputs and output appropriate error messages when validation fails.
- Use of ***REPEAT*** loop to allow user to input all ticket choices and as many as they want.
- Use of ***IF*** statements to determine the selected ticket type.
- Use of ***IF*** statements to determine whether it is 1-day booking or 2-days booking and then accordingly calculating tickets cost.
- Use of ***IF*** statement to ensure that an adult may bring up to two children.

TASK 2 – Explanation of Pseudocode:

[ALL IDENTIFIERS OF TASK 1]

```
DECLARE booking_day ← 0, no_of_days ← 0, no_of_tickets ← 0, ticket_count ← 0 AS INTEGER
DECLARE unique_booking_no ← 0, selected_ticket_type ← 0 AS INTEGER
DECLARE adult_ticket_count ← 0, child_ticket_count ← 0, senior_ticket_count ← 0
        family_ticket_count ← 0 AS INTEGER
DECLARE child_ticket_limit ← 0, group_no_of_people ← 0 AS INTEGER
DECLARE no_of_adults ← 0, no_of_seniors ← 0, no_of_children ← 0 AS INTEGER
DECLARE group_no_of_adults ← 0, group_no_of_seniors ← 0, group_no_of_children ← 0 AS INTEGER
DECLARE group_total_adults ← 0, group_total_children ← 0, group_total_seniors ← 0,
        group_total_people ← 0 AS INTEGER
DECLARE adult_cost ← 0.0, child_cost ← 0.0, senior_cost ← 0.0, family_cost ← 0.0 group_cost ← 0.0
        AS FLOAT
DECLARE lion_feeding_persons ← 0, penguin_feeding_persons ← 0, barbecue_persons ← 0 AS INTEGER
DECLARE lion_feeding_cost ← 0.0, penguin_feeding_cost ← 0.0, barbecue_cost ← 0.0 AS FLOAT
DECLARE total_attractions_cost ← 0.0, total_tickets_cost ← 0.0, total_booking_cost ← 0.0 AS FLOAT
```

Declaration of variables, constants and arrays

[ENTIRE PSEUDOCODE OF TASK 1]

```
INPUT "Kindly enter the week day for which you want to make a booking. Input 1 for Monday, 2
      for Tuesday, 3 for Wednesday, 4 for Thursday, 5 for Friday, 6 for Saturday and 7 for Sunday."
      booking_day
WHILE booking_day < 1 OR booking_day > 7
   INPUT "Wrong input. Kindly enter again.", booking_day
END WHILE
```

Input and validation of booking day using WHILE loop

```
INPUT "Kindly enter the number of days for which the booking is required. Input 1 for one day booking
      or 2 for two consecutive days booking", no_of_days
WHILE no_of_days < 1 OR no_of_days > 2
   INPUT "Wrong input. Kindly enter again.", no_of_days
END WHILE
```

Input and validation of number of days of visit using WHILE loop

```
REPEAT
```

Use of REPEAT loop to allow user to input required number of tickets

```
   INPUT "How many tickets would you like to buy?", no_of_tickets
   INPUT "What type of the ticket would you like to buy? Input 1 for Adult Ticket, 2 for Child Ticket,
         3 for Senior Ticket, 4 for Family Ticket or 5 for Group Ticket.", selected_ticket_type
   WHILE selected_ticket_type < 1 OR selected_ticket_type > 5
      INPUT "Wrong input. Kindly enter again.", selected_ticket_type
   END WHILE
```

Input and validation of selected ticket type

```
IF selected_ticket_type = 1 THEN
   adult_ticket_count ← adult_ticket_count + 1
   child_ticket_limit ← child_ticket_limit + 2
   IF no_of_days = 1 THEN
      adult_cost ← adult_ticket_count * one_day_cost[1]
   ELSE
      adult_cost ← adult_ticket_count * two_day_cost[1]
   END IF
END IF
```

Use of IF statement to determine selected ticket type and to calculate the adult ticket cost for either 1-day or 2-days booking

```
IF selected_ticket_type = 2 THEN
   IF child_ticket_limit = 0 THEN
      PRINT "The current number of adults cannot bring more children."
   ELSE
      child_ticket_limit ← child_ticket_limit – 1
      child_ticket_count ← child_ticket_count + 1
   END IF
   IF no_of_days = 1 THEN
      child_cost ← child_ticket_count * one_day_cost[2]
   ELSE
      child_cost ← child_ticket_count * two_day_cost[2]
   END IF
END IF
```

Use of IF statement to determine selected ticket type and to calculate the child ticket cost for either 1-day or 2-days booking

Use of IF statement to determine that only allowed number of children tickets are being sold in a booking

```
IF selected_ticket_type = 3 THEN
   senior_ticket_count ← senior_ticket_count + 1
   IF no_of_days = 1 THEN
      senior_cost ← senior_ticket_count * one_day_cost[3]
   ELSE
      senior_cost ← senior_ticket_count * two_day_cost[3]
   END IF
END IF
```

Use of IF statement to determine selected ticket type and to calculate the senior ticket cost for either 1-day or 2-days booking

```
IF selected_ticket_type = 4 THEN
   family_ticket_count ← family_ticket_count + 1
   INPUT "Kindly enter the number of adults. Up to two adults are allowed.", no_of_adults
   WHILE no_of_adults < 0 OR no_of_adults > 2
      INPUT "Wrong input. Kindly enter again.", no_of_adults
   END WHILE
```

Use of IF statement to determine selected ticket type

```
IF no_of_adults = 0 THEN
    INPUT "Kindly enter the number of seniors. Up to two seniors are allowed.",
          no_of_seniors
    WHILE no_of_seniors < 0 OR no_of_seniors > 2
        INPUT "Wrong input. Kindly enter again.", no_of_seniors
    END WHILE
END IF

IF no_of_adults = 1 THEN
    INPUT "Kindly enter the number of seniors. Only one senior is allowed.", no_of_seniors
    WHILE no_of_seniors < 0 OR no_of_seniors > 1
        INPUT "Wrong input. Kindly enter again.", no_of_seniors
    END WHILE
END IF

INPUT "Kindly enter the number of children. Up to three children are allowed.", no_of_children
WHILE no_of_children < 0 OR no_of_children > 3
    INPUT "Wrong input. Kindly enter again.", no_of_children
END WHILE
```

Input and validation of number of seniors and children in a family ticket using WHILE loop

```
IF no_of_days = 1 THEN
    family_cost ← family_ticket_count * one_day_cost[4]
ELSE
    family_cost ← family_ticket_count * two_day_cost[4]
END IF
END IF
```

Use of IF statement to calculate the family ticket cost for either 1-day or 2-days booking

```
IF selected_ticket_type = 5 THEN
```

Use of IF statement to determine selected ticket type

```
    INPUT "Kindly enter the total number of people in the group (six or more people).",
          group_no_of_people
    WHILE  group_no_of_people < 6
        INPUT "Wrong input. Kindly enter again.", group_no_of_people
    END WHILE
    group_total_people ← group_total_people + group_no_of_people
    REPEAT
        INPUT "Kindly enter the number of adults in the group.", group_no_of_adults
        WHILE group_no_of_adults > group_no_of_people
            INPUT "Wrong input. Kindly enter again.", group_no_of_adults
        END WHILE
```

Input and validation of number of people in a group ticket using WHILE loop

Use of REPEAT loop to allow input of required number of people

Input and validation of number of adults in a group ticket using WHILE loop

```
        child_ticket_limit ← child_ticket_limit + (group_no_of_adults * 2)
        group_no_of_people ← group_no_of_people – group_no_of_adults
        group_total_adults ← group_total_adults + group_no_of_adults

        INPUT "Kindly enter the number of children in the group.", group_no_of_children
        WHILE group_no_of_children > child_ticket_limit AND group_no_of_children >
                group_no_of_people
            INPUT "Wrong input. Kindly enter again.", group_no_of_children
        END WHILE
        child_ticket_limit ← child_ticket_limit – group_no_of_children
        group_no_of_people ← group_no_of_people – group_no_of_children
        group_total_children ← group_total_children + group_no_of_children

        INPUT "Kindly enter the number of seniors in the group.", group_no_of_seniors
        WHILE group_no_of_seniors > group_no_of_people
            INPUT "Wrong input. Kindly enter again.", group_no_of_seniors
        END WHILE
        group_no_of_people ← group_no_of_people – group_no_of_seniors
        group_total_seniors ← group_total_seniors + group_no_of_seniors

    UNTIL group_no_of_people = 0
```

Input and validation of number of children and number of seniors in a family ticket using WHILE loop

```
    IF no_of_days = 1 THEN
        group_cost ← group_total_people * one_day_cost[5]
    ELSE
        group_cost ← group_total_people * two_day_cost[5]
    END IF
END IF
```

Use of IF statement to calculate the group ticket cost for either 1-day or 2-days booking

```
ticket_count ← ticket_count + 1

UNTIL ticket_count = no_of_tickets
```

Ticket counter being updated to ensure required number of tickets are input

```
PRINT "Kindly enter the following details if you want to have extra attractions included:"
INPUT "Kindly enter the number of persons who want to feed a lion: ", lion_feeding_persons
lion_feeding_cost ← lion_feeding_persons * extra_attraction_cost[1]
INPUT "Kindly enter the number of persons who want to feed a penguin: ", penguin_feeding_persons
penguin_feeding_cost ← penguin_feeding_persons * extra_attraction_cost[2]
```

Input of number of people who want to do lion or penguin feeding (extra attractions)

```
IF no_of_days = 2 THEN
    INPUT "Kindly enter the number of persons who want to do an evening barbecue", barbecue_persons
    barbecue_cost ← barbecue_persons * extra_attraction_cost[3]
END IF
```

Use of IF statement to check if booking is eligible for evening barbecue (extra attraction)

```
unique_booking_no ← unique_booking_no + 1
```

Allocating a unique booking number

```
total_attractions_cost ← lion_feeding_cost + penguin_feeding_cost + barbecue_cost
total_tickets_cost ← adult_cost + child_cost + senior_cost + family_cost + group_cost

total_booking_cost ← total_attractions_cost + total_tickets_cost
```

Totaling the tickets cost and extra attractions cost and then storing the price in total booking cost

```
PRINT "The following are your booking details:"

PRINT "The unique booking number is: ", unique_booking_no
PRINT "The booking is made for the following week day: ", booking_day
PRINT "The booking is only valid for following number of day(s): ", no_of_days
PRINT "The following number of ticket(s) were bought: ", no_of_tickets
PRINT "The following number of adult ticket(s) were bought: ", adult_ticket_count
PRINT "The following number of child ticket(s) were bought: ", child_ticket_count
PRINT "The following number of senior ticket(s) were bought: ", senior_ticket_count
PRINT "The following number of family ticket(s) were bought: ", family_ticket_count
PRINT "The following number of people are part of group ticket(s): ", group_total_people
PRINT "The following number of person(s) want to feed a lion: ", lion_feeding_persons
PRINT "The following number of person(s) want to feed a penguin: ", penguin_feeding_persons
PRINT "The following number of person(s) want to do an evening barbecue: ", barbecue_persons
PRINT "The following is the total cost for extra attractions: ", total_attractions_cost
PRINT "The following is the total cost for tickets bought: ", total_tickets_cost
PRINT "The following is the grand total cost for complete booking: ", total_booking_cost
```

Required output of TASK 2

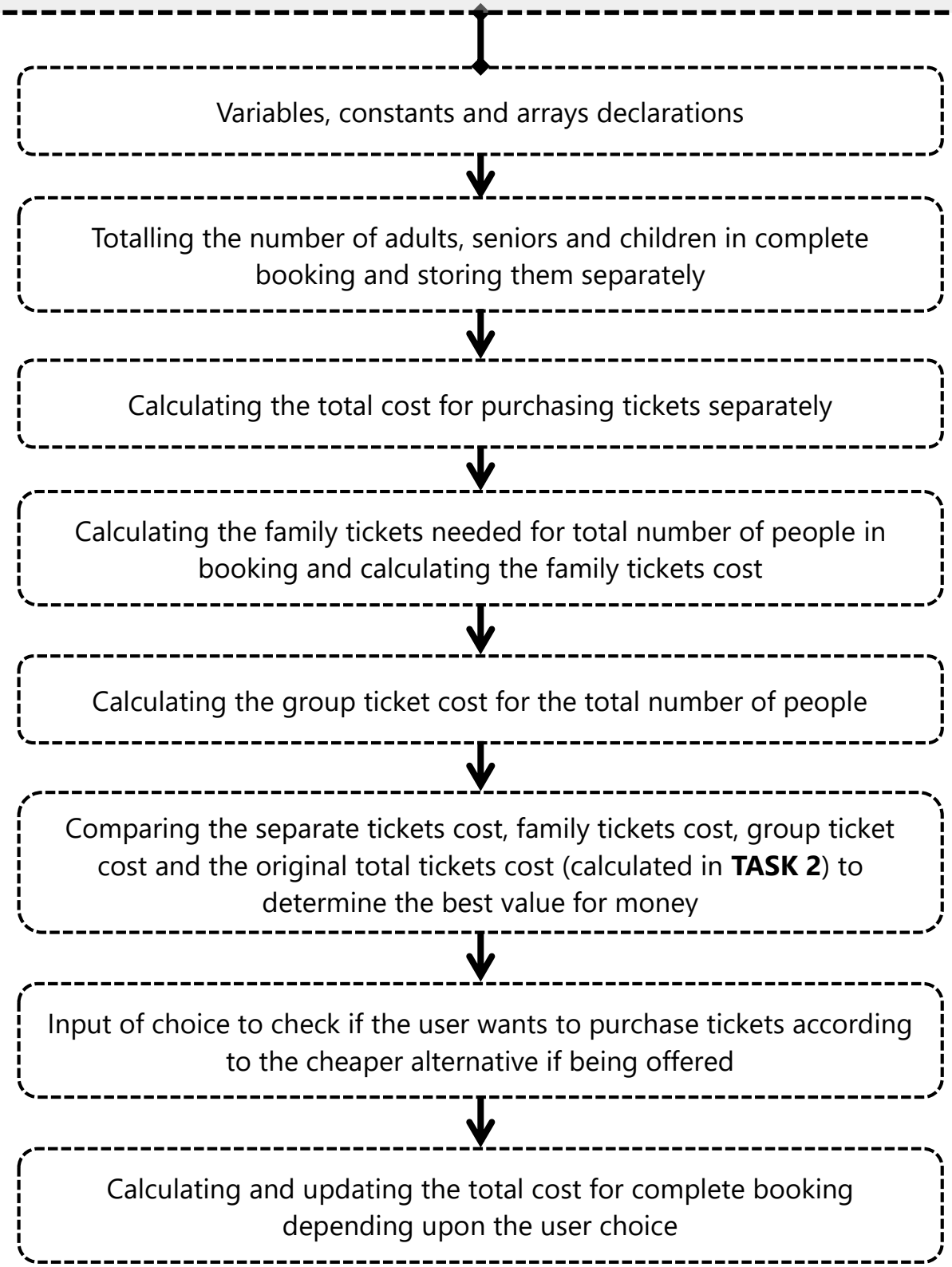
TASK 2 – Expected Questions:

1. State two variables you used for Task 2. State the data type and purpose of the variables.
2. Describe the data structures you have used in Task 2. Include the name(s), data type, sample data and usage for each structure.
3. Write an algorithm for Task 2, using either Pseudocode, programming statements or a flowchart. You should assume that Task 1 has already been completed.
4. Write an algorithm to complete Task 2 without including any error prompts, using either Pseudocode, programming statements or a flowchart. You should assume that Task 1 has already been completed.
5. Explain how your program completes/performs Task 2. Any programming statements used in your answer must be fully explained.
6. Explain how you calculated the total cost for the booking (part of Task 2). You can include Pseudocode or programming statements as part of your explanation.
7. Explain how you ensured that the number of children do not exceed the allowed limit. You can include Pseudocode or programming statements as part of your explanation.
8. Explain how you ensured that the conditions of a family ticket are being satisfied. You can include Pseudocode or programming statements as part of your explanation.
9. Explain how you validated any two inputs used in Task 2. State one valid and one invalid input to test your validation methods (valid and invalid test data). You can include Pseudocode or programming statements as part of your explanation.
10. Write an algorithm for Task 2, using either Pseudocode, programming statements or a flowchart. Change the algorithm to ensure that one user can only buy 5 tickets. You should assume that Task 1 has already been completed.
11. Write an algorithm for Task 2, using either Pseudocode, programming statements or a flowchart. Change the algorithm to ensure that evening barbecue is made available for one-day tickets as well. You should assume that Task 1 has already been completed.
12. Write an algorithm for Task 2, using either Pseudocode, programming statements or a flowchart. Change the algorithm to ensure that only two-days bookings are allowed. You should assume that Task 1 has already been completed.
13. Comment on the efficiency of your code for Task 2.

Concept and understanding of TASK 3:

Task 3 – ensuring each booking is the best value

Check that the total for each booking gives the best value and offer an alternative if this is **not** the case. For example, buying two family tickets is better than a group ticket for a group of 10 that includes four adults and six children.



Explanation of Algorithm of TASK 3:

In this task, we have to check that the total cost for each booking is the cheapest possible cost. If not, then the newly calculated cheapest possible cost would be offered as an alternative to the user so that they can save money and choose that instead.

> Check that the total for each booking gives the best value and offer an alternative if this is **not** the case. For example, buying two family tickets is better than a group ticket for a group of 10 that includes four adults and six children.

Firstly, we will separately total the number of adults, children, seniors and total people involved in the booking and then store them in the following variables:

- total_adults
- total_seniors
- total_children
- total_people

The following piece of code would be used with concept of totaling:

```
total_adults ← adult_ticket_count + no_of_adults + group_total_adults
total_seniors ← senior_ticket_count + no_of_seniors + group_total_seniors
total_children ← child_ticket_count + no_of_children + group_total_children
total_people ← total_adults + total_seniors + total_children
```

→ All variables being totaled in first 3 statements are from **TASK 2**.

Three different total costs for buying tickets would be calculated:

- Calculating the total cost for buying tickets separately
- Calculating the number of family tickets needed to accommodate total people and their costs
- Calculating the total cost for buying group ticket to accommodate total people

(i) Separate Tickets:

IF statement would be used to determine whether it is 1-day or 2-days booking and then accordingly the individual total number of adults, seniors and children would be multiplied with their individual ticket costs:

```
IF no_of_days = 1 THEN
    separate_tickets_cost ← (total_adults * one_day_cost[1]) + (total_seniors * one_day_cost[2]) +
                            (total_children * one_day_cost[3])
ELSE
    separate_tickets_cost ← (total_adults * two_day_cost[1]) + (total_seniors * two_day_cost[2]) +
                            (total_children * two_day_cost[3])
END IF
```

This would give the total cost for buying tickets separately from the following 3 ticket types:

- One adult
- One child (an adult may bring up to two children)
- One senior

The total cost would then be stored in the variable **separate_tickets_cost**.

(ii) Family Tickets:

First, the total number of adults and seniors would be added together as the family ticket type deals with adults and seniors as one (up to two adults **OR** two seniors).

The total number would be stored in variable **adults_and_seniors**.

IF statement would be used to determine how many family tickets are needed to accommodate the number of **adults_and_seniors**. Moreover, **DIV** and **MOD** functions will also be used in the process:

```
IF adults_and_seniors * 1.5 >= total_children THEN
    family_tickets_needed ← (adults_and_seniors DIV 2) + (adults_and_seniors MOD 2)
```

Family ticket condition: up to two adults or seniors, and three children

So the limit for family ticket is 2 adults/seniors and 3 children. Therefore **3/2 = 1.5** and so the total number of **adults_and_seniors** would be multiplied by **1.5** and compared with **total_children.**

If the total number of **adults_and_seniors * 1.5** is greater than or equal to **total_children**, only then would the condition be satisfied and buying family tickets would be an option.

For example, if there are 4 **adults_and_seniors** and 8 **total_children** then condition would fail as:

- (4 * 1.5 = 6) which is less than total children (8)
- one family ticket allowed 2 adults/seniors and 3 children
- if there are 4 adults/seniors than 6 children would be allowed (4 * 1.5) (two family tickets)
- so technically it is impossible to accommodate 8 children in the family ticket type.

Similarly, if there are 6 **adults_and_seniors** and 8 **total_children** then condition would be satisfied as:

- (6 * 1.5 = 9) which is greater than total children (8)
- one family ticket allowed 2 adults/seniors and 3 children
- if there are 6 adults/seniors than 9 children would be allowed (6 * 1.5) (three family tickets)

DIV/MOD Function:

The ***IF*** statement would have determined if buying family tickets is an option (according to checking the family ticket condition of 2 adults/seniors and 3 children).

If family ticket is checked to be a valid option, only then **DIV** and **MOD** functions will be used to calculate **the number of family tickets needed** to accommodate the adults/seniors and children:

- DIV gives us the integer value which is quotient.
- MOD gives us the remainder.

We are dividing **adults_and_seniors** by 2 because for every 1 family ticket, 2 adults/seniors are allowed. Hence if there are 14 adults/seniors then 14 divided by 2 is 7 which is the number of family tickets needed to accommodate 14 adults/seniors. 7 family tickets can accommodate 14 adults/seniors (as 1 ticket allows up to 2 adults/seniors)

For example:

- 6 DIV 2 = 3 (integer value)
- 9 DIV 2 = 4 (integer value)
- 6 MOD 2 = 0 (no remainder)
- 9 MOD 2 = 1 (remainder)

The value of quotient (DIV function) and remainder (MOD function) would be added together to calculate the number of family tickets needed to accommodate the amount of adults/seniors.

For example, if **adults_and_seniors** = 11:

- family_tickets_needed ← (adults_and_seniors **DIV** 2) **+** (adults_and_seniors **MOD** 2)
- family_tickets_needed ← (11 **DIV** 2) **+** (11 **MOD** 2)
- family_tickets_needed ← (5) **+** (1) ← **Remainder**
- family_tickets_needed ← 5 + 1
- family_tickets_needed ← 6

Therefore, **6 family tickets** are needed to accommodate 11 adults/seniors.

You can also manually calculate this to understand the concept of approaching **DIV** and **MOD** functions in the algorithm.

Manual Calculation:

- 2 adults/seniors = 1 ticket
- 11 adults/seniors = 11/2 = 5.5 tickets
- Now since you know that a half ticket cannot be bought therefore it will be rounded to the nearest integer
- Hence, 5.5 rounded → **6 family tickets**

Another ***IF*** statement would be used to determine whether it is 1-day or 2-days booking and then accordingly the total number of family tickets needed would be multiplied with the family ticket cost:

```
IF adults_and_seniors * 1.5 >= total_children THEN
    family_tickets_needed ← (adults_and_seniors DIV 2) + (adults_and_seniors MOD 2)
    IF no_of_days = 1 THEN
        family_tickets_cost ← family_tickets_needed * one_day_cost[4]
    ELSE
        family_tickets_cost ← family_tickets_needed * two_day_cost[4]
    END IF
END IF
```

The total cost would then be stored in the variable **family_tickets_cost**.

(iii) Group Tickets:

IF statement would be used to determine if the total number of people is greater than 6:

Group ticket condition: groups of six people or more

If the condition is TRUE, then another ***IF*** statement would be used to determine whether it is 1-day or 2-days booking and then accordingly the total number of people would be multiplied with the group ticket cost:

```
IF total_people >= 6 THEN
    IF no_of_days = 1 THEN
        group_tickets_cost ← total_people * one_day_cost[5]
    ELSE
        group_tickets_cost ← total_people * two_day_cost[5]
    END IF
END IF
```

The total cost would then be stored in the variable **group_tickets_cost**.

The three different total costs for buying tickets have been calculated and then the comparisons would be made between four total costs:

- separate_tickets_cost
- family_tickets_cost
- group_tickets_cost
- total_tickets_cost (which is the original cost calculated according to user inputs for selection of tickets)

Multiple ***IF*** statements would be used to make comparisons between all 4 total costs. If any cost is cheaper than all others, it would be stored in the variable **cheapest_cost** and the user would be shown the cheapest total cost alongside the ticketing method used for finding this best value for money:

```
IF separate_tickets_cost < family_tickets_cost AND separate_tickets_cost < group_tickets_cost
   AND separate_tickets_cost < total_tickets_cost THEN
    cheapest_cost ← separate_tickets_cost
    PRINT "The best value for money is to buy separate tickets for each adult, child and senior"
    PRINT "The cheapest total cost for tickets would be: ", cheapest_cost
ELSE IF family_tickets_cost < separate_tickets_cost AND family_tickets_cost < group_tickets_cost
    AND family_tickets_cost < total_tickets_cost THEN
    cheapest_cost ← family_tickets_cost
    PRINT "The best value for money is to buy the following number of family tickets: ",
          family_tickets_needed
    PRINT "The cheapest total cost for tickets would be: ", cheapest_cost
ELSE IF group_tickets_cost < separate_tickets_cost AND group_tickets_cost < family_tickets_cost
    AND group_tickets_cost < total_tickets_cost THEN
    cheapest_cost ← group_tickets_cost
    PRINT "The best value for money is to buy the group ticket for following number of people: ",
          total_people
    PRINT "The cheapest total cost for tickets would be: ", cheapest_cost
```

If none of the 3 calculated total costs is cheaper than, the user would be shown the original tickets cost calculated in **TASK 2** according to user inputs for selection of every ticket(s):

```
ELSE
    PRINT "The best value for money is to buy the tickets exactly the way you have bought"
    PRINT "The cheapest total cost for tickets is: ", total_tickets_cost
    PRINT "The following is the total cost for extra attractions: ", total_attractions_cost
    PRINT "The following is the grand total cost for complete booking: ", total_booking_cost
```

Then an ***IF*** statement would be used to check if there is any cheaper total as compared to the original total cost. If the variable **cheapest_cost** is not zero and so has some value stored, then it means that a cheaper total exists and so the user would be asked to INPUT a choice regarding if they want to purchase tickets according to the cheaper alternative:

```
IF cheapest_cost <> 0 THEN
    INPUT "Would you like to purchase tickets according to the cheaper alternative being offered?
          Y or N?", choice
  WHILE choice <> "Y" OR choice <> "N"
    INPUT "Wrong input. Kindly enter again", choice
  END WHILE
END IF
```

Another ***IF*** statement would be used to determine choice. If the user has chosen the cheaper alternative then the **total_booking_cost (TASK 2)** would be updated by adding the new total tickets cost stored in **cheapest_cost** and **total_attractions_cost (TASK 2)**.

The user would then simply be shown their updated booking cost:

```
IF choice = "Y" THEN
  total_booking_cost ← total_attractions_cost + cheapest_cost
  PRINT "You selected the alternative total for the booking which gives the best value: "
  PRINT "The following is the total cost for extra attractions: ", total_attractions_cost
  PRINT "The following is the updated cheapest total cost for tickets bought: ", cheapest_cost
  PRINT "The following is the updated cheapest grand total cost for complete booking: ",
        total_booking_cost
END IF
```

If the user has denied the cheaper alternative and chosen their original tickets and booking cost then the user would simply be shown their unchanged tickets cost and booking cost:

```
IF choice = "N" THEN
  PRINT "You did not select the alternative total for the booking which gives the best value: "
  PRINT "The following is the total cost for extra attractions: ", total_attractions_cost
  PRINT "The following is the total cost for tickets bought: ", total_tickets_cost
  PRINT "The following is the grand total cost for complete booking: ", total_booking_cost
END IF
```

TASK 3 – Pseudocode:

BEGIN

```
DECLARE total_adults ← 0, total_seniors ← 0, total_children ← 0, total_people ← 0 AS INTEGER
DECLARE adults_and_seniors ← 0, family_tickets_needed ← 0 AS INTEGER
DECLARE separate_tickets_cost ← 0.0, family_tickets_cost ← 0.0, group_tickets_cost ← 0.0 AS FLOAT
DECLARE cheapest_cost ← 0.0 AS FLOAT
DECLARE choice ← "" AS STRING

total_adults ← adult_ticket_count + no_of_adults + group_total_adults
total_seniors ← senior_ticket_count + no_of_seniors + group_total_seniors
total_children ← child_ticket_count + no_of_children + group_total_children
total_people ← total_adults + total_seniors + total_children

IF no_of_days = 1 THEN
    separate_tickets_cost ← (total_adults * one_day_cost[1]) + (total_seniors * one_day_cost[2]) +
                            (total_children * one_day_cost[3])
ELSE
    separate_tickets_cost ← (total_adults * two_day_cost[1]) + (total_seniors * two_day_cost[2]) +
                            (total_children * two_day_cost[3])
END IF

adults_and_seniors ← total_adults + total_seniors

IF adults_and_seniors * 1.5 >= total_children THEN
    family_tickets_needed ← (adults_and_seniors DIV 2) + (adults_and_seniors MOD 2)
    IF no_of_days = 1 THEN
        family_tickets_cost ← family_tickets_needed * one_day_cost[4]
    ELSE
        family_tickets_cost ← family_tickets_needed * two_day_cost[4]
    END IF
END IF

IF total_people >= 6 THEN
    IF no_of_days = 1 THEN
        group_tickets_cost ← total_people * one_day_cost[5]
    ELSE
        group_tickets_cost ← total_people * two_day_cost[5]
    END IF
END IF
```

```
IF separate_tickets_cost < family_tickets_cost AND separate_tickets_cost < group_tickets_cost
   AND separate_tickets_cost < total_tickets_cost THEN

    cheapest_cost ← separate_tickets_cost

    PRINT "The best value for money is to buy separate tickets for each adult, child and senior"
    PRINT "The cheapest total cost for tickets would be: ", cheapest_cost

ELSE IF family_tickets_cost < separate_tickets_cost AND family_tickets_cost < group_tickets_cost
    AND family_tickets_cost < total_tickets_cost THEN

    cheapest_cost ← family_tickets_cost

    PRINT "The best value for money is to buy the following number of family tickets: ",
          family_tickets_needed
    PRINT "The cheapest total cost for tickets would be: ", cheapest_cost

ELSE IF group_tickets_cost < separate_tickets_cost AND group_tickets_cost < family_tickets_cost
    AND group_tickets_cost < total_tickets_cost THEN

    cheapest_cost ← group_tickets_cost

    PRINT "The best value for money is to buy the group ticket for following number of people: ",
          total_people
    PRINT "The cheapest total cost for tickets would be: ", cheapest_cost

ELSE

    PRINT "The best value for money is to buy the tickets exactly the way you have bought"
    PRINT "The cheapest total cost for tickets is: ", total_tickets_cost
    PRINT "The following is the total cost for extra attractions: ", total_attractions_cost
    PRINT "The following is the grand total cost for complete booking: ", total_booking_cost

END IF
END IF
END IF


IF cheapest_cost <> 0 THEN

      INPUT "Would you like to purchase tickets according to the cheaper alternative being offered?
             Y or N?", choice
   WHILE choice <> "Y" OR choice <> "N"
      INPUT "Wrong input. Kindly enter again", choice
   END WHILE

END IF
```

```
IF choice = "Y" THEN
    total_booking_cost ← total_attractions_cost + cheapest_cost
    PRINT "You selected the alternative total for the booking which gives the best value: "
    PRINT "The following is the total cost for extra attractions: ", total_attractions_cost
    PRINT "The following is the updated cheapest total cost for tickets bought: ", cheapest_cost
    PRINT "The following is the updated cheapest grand total cost for complete booking: ",
          total_booking_cost
END IF

IF choice = "N" THEN
    PRINT "You did not select the alternative total for the booking which gives the best value: "
    PRINT "The following is the total cost for extra attractions: ", total_attractions_cost
    PRINT "The following is the total cost for tickets bought: ", total_tickets_cost
    PRINT "The following is the grand total cost for complete booking: ", total_booking_cost
END IF

END
```

TASK 3 – Efficiency:

- Use of concept of ***TOTALLING*** to calculate the total adults, seniors, children and people.
- Use of ***IF*** statements to determine whether it is 1-day booking or 2-days booking and then accordingly calculating tickets cost.
- Use of ***IF*** statements to compare different costs and then calculate the cheapest total cost.
- Use of ***IF*** statements to input user choice regarding cheaper alternative and then accordingly displaying the total booking cost.
- Use of ***DIV*** and ***MOD*** function to calculate the number of family tickets that the user can purchase to accommodate all people.
- Use of ***WHILE*** loop to validate all user inputs and output appropriate error messages when validation fails.

TASK 3 – Explanation of Pseudocode:

```
DECLARE total_adults ← 0, total_seniors ← 0, total_children ← 0, total_people ← 0 AS INTEGER
DECLARE adults_and_seniors ← 0, family_tickets_needed ← 0 AS INTEGER
DECLARE separate_tickets_cost ← 0.0, family_tickets_cost ← 0.0, group_tickets_cost ← 0.0 AS FLOAT
DECLARE cheapest_cost ← 0.0 AS FLOAT
DECLARE choice ← "" AS STRING
```

Declaration of variables, constants and arrays

```
total_adults ← adult_ticket_count + no_of_adults + group_total_adults
total_seniors ← senior_ticket_count + no_of_seniors + group_total_seniors
total_children ← child_ticket_count + no_of_children + group_total_children
total_people ← total_adults + total_seniors + total_children
```

Totaling the number of adults, seniors, children and people

```
IF no_of_days = 1 THEN
    separate_tickets_cost ← (total_adults * one_day_cost[1]) + (total_seniors * one_day_cost[2]) +
                            (total_children * one_day_cost[3])
ELSE
    separate_tickets_cost ← (total_adults * two_day_cost[1]) + (total_seniors * two_day_cost[2]) +
                            (total_children * two_day_cost[3])
END IF
```

Use of IF statement to calculate the separate tickets cost for either 1-day or 2-days booking

```
adults_and_seniors ← total_adults + total_seniors
```

```
IF adults_and_seniors * 1.5 >= total_children THEN
    family_tickets_needed ← (adults_and_seniors DIV 2) + (adults_and_seniors MOD 2)
    IF no_of_days = 1 THEN
        family_tickets_cost ← family_tickets_needed * one_day_cost[4]
    ELSE
        family_tickets_cost ← family_tickets_needed * two_day_cost[4]
    END IF
END IF
```

Use of IF statement and DIV and MOD functions to calculate the number of family tickets needed and then another IF statement to calculate the family tickets cost for either 1-day or 2-days booking

```
IF total_people >= 6 THEN
    IF no_of_days = 1 THEN
        group_tickets_cost ← total_people * one_day_cost[5]
    ELSE
        group_tickets_cost ← total_people * two_day_cost[5]
    END IF
END IF
```

Use of IF statement to check that the condition for group ticketing is TRUE and then another IF statement to calculate the group tickets cost for either 1-day or 2-days booking

```
IF separate_tickets_cost < family_tickets_cost AND separate_tickets_cost < group_tickets_cost
   AND separate_tickets_cost < total_tickets_cost THEN
   cheapest_cost ← separate_tickets_cost
   PRINT "The best value for money is to buy separate tickets for each adult, child and senior"
   PRINT "The cheapest total cost for tickets would be: ", cheapest_cost
ELSE IF family_tickets_cost < separate_tickets_cost AND family_tickets_cost < group_tickets_cost
    AND family_tickets_cost < total_tickets_cost THEN
   cheapest_cost ← family_tickets_cost
   PRINT "The best value for money is to buy the following number of family tickets: ",
         family_tickets_needed
   PRINT "The cheapest total cost for tickets would be: ", cheapest_cost
ELSE IF group_tickets_cost < separate_tickets_cost AND group_tickets_cost < family_tickets_cost
   AND group_tickets_cost < total_tickets_cost THEN
   cheapest_cost ← group_tickets_cost
   PRINT "The best value for money is to buy the group ticket for following number of people: ",
         total_people
   PRINT "The cheapest total cost for tickets would be: ", cheapest_cost
ELSE
   PRINT "The best value for money is to buy the tickets exactly the way you have bought"
   PRINT "The cheapest total cost for tickets is: ", total_tickets_cost
   PRINT "The following is the total cost for extra attractions: ", total_attractions_cost
   PRINT "The following is the grand total cost for complete booking: ", total_booking_cost
END IF
END IF
END IF
```

Use of multiple IF statements to make comparisons between different calculated ticket costs and then determining the cheapest total cost

```
IF cheapest_cost <> 0 THEN
    INPUT "Would you like to purchase tickets according to the cheaper alternative being offered?
          Y or N?", choice
   WHILE choice <> "Y" OR choice <> "N"
      INPUT "Wrong input. Kindly enter again", choice
   END WHILE
END IF
```

Input and validation of user choice regarding cheaper alternative being offered

```
IF choice = "Y" THEN
    total_booking_cost ← total_attractions_cost + cheapest_cost
    PRINT "You selected the alternative total for the booking which gives the best value: "
    PRINT "The following is the total cost for extra attractions: ", total_attractions_cost
    PRINT "The following is the updated cheapest total cost for tickets bought: ", cheapest_cost
    PRINT "The following is the updated cheapest grand total cost for complete booking: ",
          total_booking_cost
END IF
```

Use of IF statement to determine user choice and then accordingly updating the total booking cost and giving the OUTPUT of updated total costs

```
IF choice = "N" THEN
    PRINT "You did not select the alternative total for the booking which gives the best value: "
    PRINT "The following is the total cost for extra attractions: ", total_attractions_cost
    PRINT "The following is the total cost for tickets bought: ", total_tickets_cost
    PRINT "The following is the grand total cost for complete booking: ", total_booking_cost
END IF
```

Use of IF statement to determine user choice and then giving the OUTPUT of unchanged total tickets and booking cost

TASK 3 – Expected Questions:

1. State two variables you used for Task 3. State the data type and purpose of the variables.
2. Describe the data structures you have used in Task 3. Include the name(s), data type, sample data and usage for each structure.
3. Write an algorithm for Task 3, using either Pseudocode, programming statements or a flowchart. You should assume that Task 1 and Task 2 have already been completed.
4. Write an algorithm to complete Task 3 without including any output messages, using either Pseudocode, programming statements or a flowchart. You should assume that Task 1 and Task 2 have already been completed.
5. Explain how your program completes/performs Task 3. Any programming statements used in your answer must be fully explained.
6. Explain how you calculated the alternative best value for money (part of Task 3). You can include Pseudocode or programming statements as part of your explanation.
7. Comment on the efficiency of your code for Task 3.