# SL Unit 4 – Problem Solving

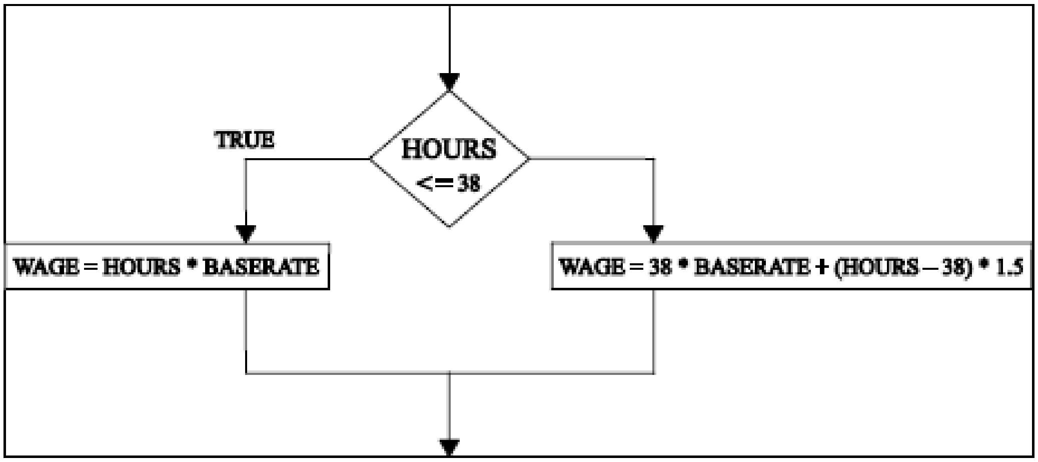
## Quiz 1

| Question 1 | | | |
|---|---|---|---|
| Objectives: | 4.2.6 | Exam Reference: | Nov-14 7 |

When the wages for company employees are calculated, all hours above 38 are paid at the overtime rate of 1.5 times the base rate.

Construct a flowchart that represents this algorithm. [3]

*Award marks as follows, up to [3* ***marks*** *max].*
*Award [1* ***mark****] for decision structure.*
*Award [1* ***mark****] for correct condition.*
*Award [1* ***mark****] for correct expression for calculation of wage when there are no overtime hours.*
*Award [1* ***mark****] for correct calculation of wage when there are overtime hours.*

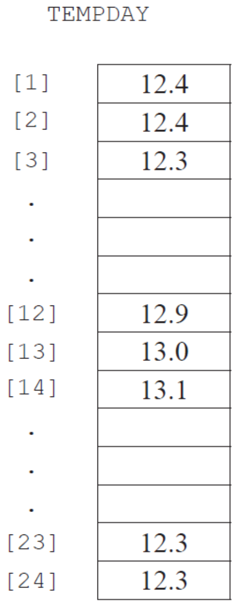


C

# Question 2

| | | | |
|---|---|---|---|
| Objectives: | 4.2.6 | Exam Reference: | Nov-14 10.a.b.c |

1. The temperature of a lake for one day is recorded every hour and data is stored in a one-dimensional array named *TEMPDAY*.



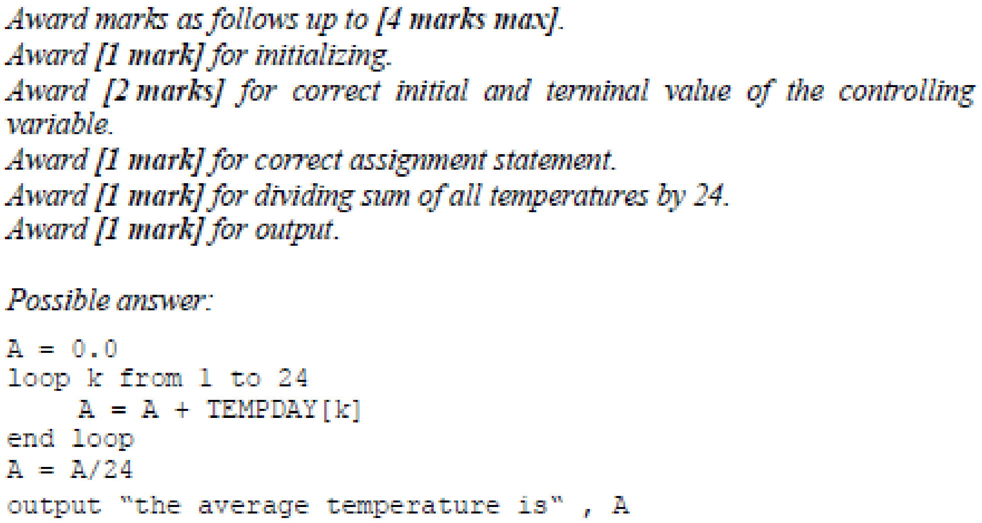
TEMPDAY

| | |
|---|---|
| [1] | 12.4 |
| [2] | 12.4 |
| [3] | 12.3 |
| . | |
| . | |
| . | |
| [12] | 12.9 |
| [13] | 13.0 |
| [14] | 13.1 |
| . | |
| . | |
| . | |
| [23] | 12.3 |
| [24] | 12.3 |

(a) State the temperature of the lake at noon. [1]

12.9

(b) Construct an algorithm that will calculate and output the average temperature. [4 ]

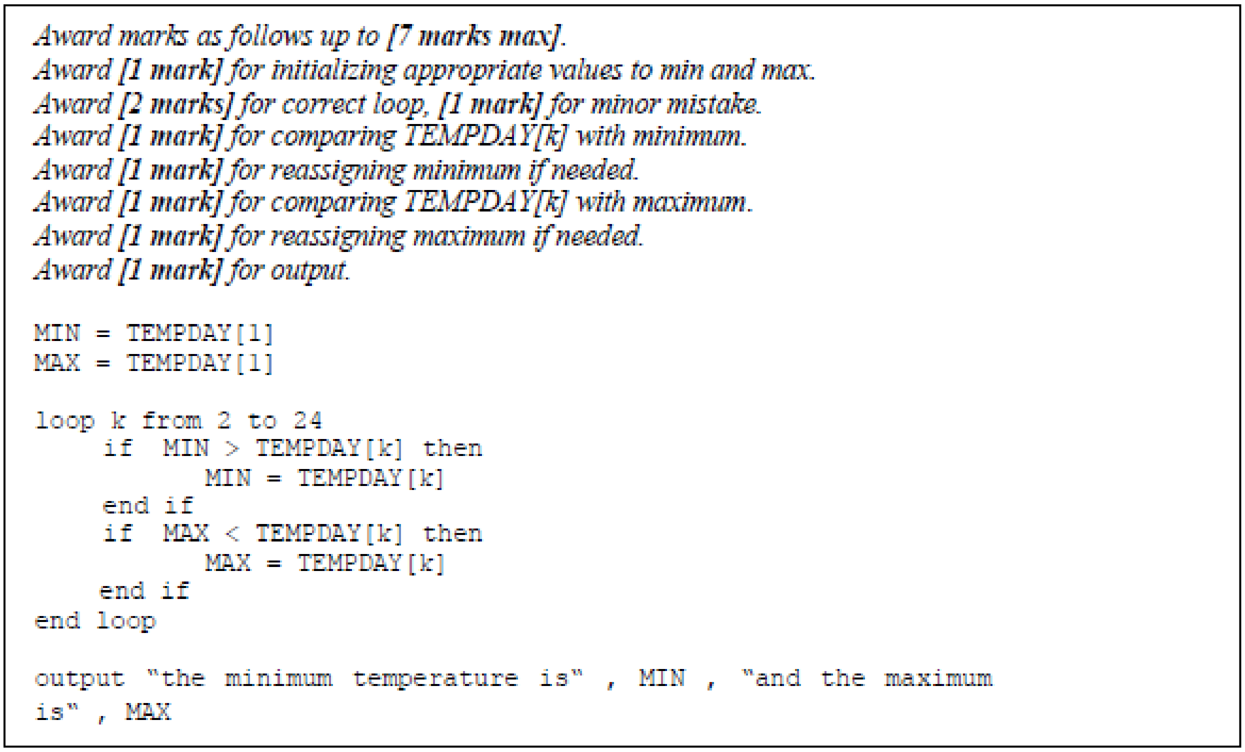


*Award marks as follows up to [4 marks max].*
*Award [1 mark] for initializing.*
*Award [2 marks] for correct initial and terminal value of the controlling variable.*
*Award [1 mark] for correct assignment statement.*
*Award [1 mark] for dividing sum of all temperatures by 24.*
*Award [1 mark] for output.*

*Possible answer:*

```
A = 0.0
loop k from 1 to 24
     A = A + TEMPDAY[k]
end loop
A = A/24
output "the average temperature is" , A
```

Construct an algorithm to find and output the minimum and maximum temperatures for the day. [7]



*Award marks as follows up to **[7 marks max]**.*
*Award **[1 mark]** for initializing appropriate values to min and max.*
*Award **[2 marks]** for correct loop, **[1 mark]** for minor mistake.*
*Award **[1 mark]** for comparing TEMPDAY[k] with minimum.*
*Award **[1 mark]** for reassigning minimum if needed.*
*Award **[1 mark]** for comparing TEMPDAY[k] with maximum.*
*Award **[1 mark]** for reassigning maximum if needed.*
*Award **[1 mark]** for output.*

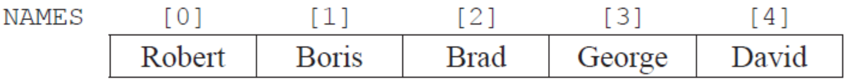
```
MIN = TEMPDAY[1]
MAX = TEMPDAY[1]

loop k from 2 to 24
     if  MIN > TEMPDAY[k] then
            MIN = TEMPDAY[k]
     end if
     if  MAX < TEMPDAY[k] then
            MAX = TEMPDAY[k]
     end if
end loop

output "the minimum temperature is" , MIN , "and the maximum
is" , MAX
```

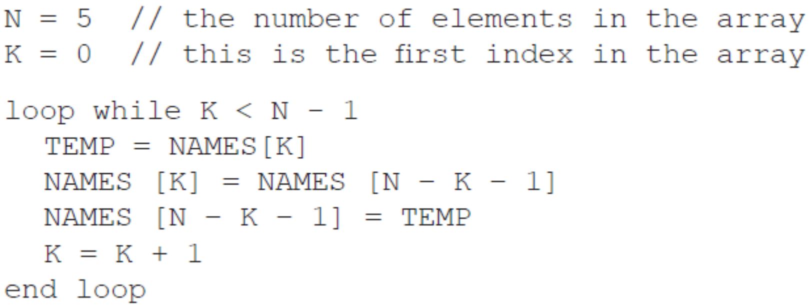
| Question 3 | | | |
|---|---|---|---|
| Objectives: | 4.2.5 | Exam Reference: | Nov-14 8 |

Consider the following array



| NAMES | [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|---|
| | Robert | Boris | Brad | George | David |

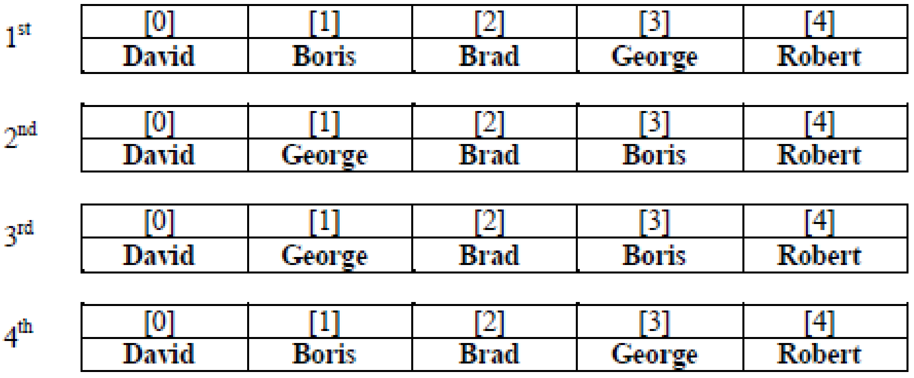
and the following algorithm, which is constructed to reverse the contents of the array *NAMES*.



```
N = 5  // the number of elements in the array
K = 0  // this is the first index in the array

loop while K < N - 1
  TEMP = NAMES[K]
  NAMES [K] = NAMES [N - K - 1]
  NAMES [N - K - 1] = TEMP
  K = K + 1
end loop
```

Trace the algorithm, showing the contents of the array after each execution of the loop. [2]



1st

| [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|
| **David** | **Boris** | **Brad** | **George** | **Robert** |

2nd

| [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|
| **David** | **George** | **Brad** | **Boris** | **Robert** |

3rd

| [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|
| **David** | **George** | **Brad** | **Boris** | **Robert** |

4th

| [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|
| **David** | **Boris** | **Brad** | **George** | **Robert** |

(a) Identify the type of error that occurs. [1]

Logic error;

(b) *Outline why the error occurs and how it could be corrected.* [2]

*Award* ***[1 mark]*** *for stating a possible cause of error.*
Loop executes too many times;
Terminating value for controlling variable was not correctly set;

*Award* ***[1 mark]*** *for stating a possible solution.*
Condition should be changed to k = n div 2;

# Question 4

| Objectives: | 4.2.1, 4.2.3, 4.2.6 | Exam Reference: | May-16 13 |
|---|---|---|---|

A local charity organizes a half-marathon to raise money. The rules to participate in the half-marathon are as follows:

- The organizers limit the total number of participants to 450
- Participants belong to a team and each team must have at least three and at most five
- participants
- Each participant registers for the event independently from the other members of their
- team, and they all declare their team name when registering
- For scoring, the team's final time is the sum of the times of its three fastest participants. Participants that do not cross the finishing line within 2 hours after the start, are assigned a default time of 1000 minutes. The **winning team** is the team with the smallest sum total.

During registration, an array, *PARTICIPANTS*, with 450 positions is used to hold the abbreviated team names that are declared by each participant. Simultaneously, a collection *TNAMES* is generated: any new team name that is declared is added to the collection.

(a) State the minimum size of *TNAMES* to ensure the names of all potential teams can be stored. [1]

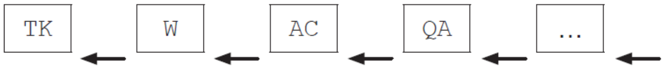
150 (= 450/3);

Part of the array *PARTICIPANTS* is shown below, where, for example, the first participant declared that they are part of team *TK*. The initial part of the collection *TNAMES* is also shown, with arrows indicating the direction of growth.

PARTICIPANTS

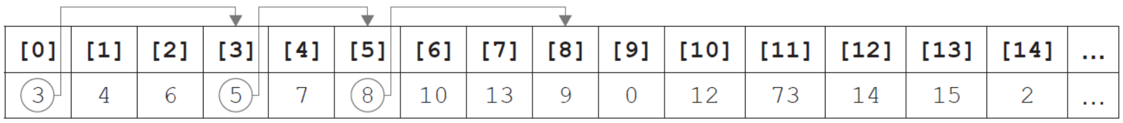
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] | [14] | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TK | W | AC | TK | W | TK | AC | W | TK | TK | AC | QA | AC | W | AC | … |

TNAMES



Both *PARTICIPANTS* and *TNAMES* are used to construct the array, *TEAM*, that groups all participants who belong to the same team. Part of the array *TEAM* is shown below.

TEAM



| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] | [14] | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 4 | 6 | 5 | 7 | 8 | 10 | 13 | 9 | 0 | 12 | 73 | 14 | 15 | 2 | ... |

In *TEAM*, each element is related to one other index in the array, shown by the arrows on the above diagram. This relation will eventually form a closed path (for this example 0, 3, 5, 8, 9 and back to 0). The relation reflects the information in *PARTICIPANTS*, by grouping people who declared the same team name during registration.

Hence, participants 0, 3, 5, 8 and 9 are on the same team and, from *PARTICIPANTS*, that team is *TK*.

(b) Identify the position in *PARTICIPANTS* of the second participant that registered for team *QA*. [1]

73;
***Note:*** *Accept* `PARTICIPANTS[73]`*. Do not accept* `TEAM[11]` *because the question asks specifically for the array* `PARTICIPANTS`*.*

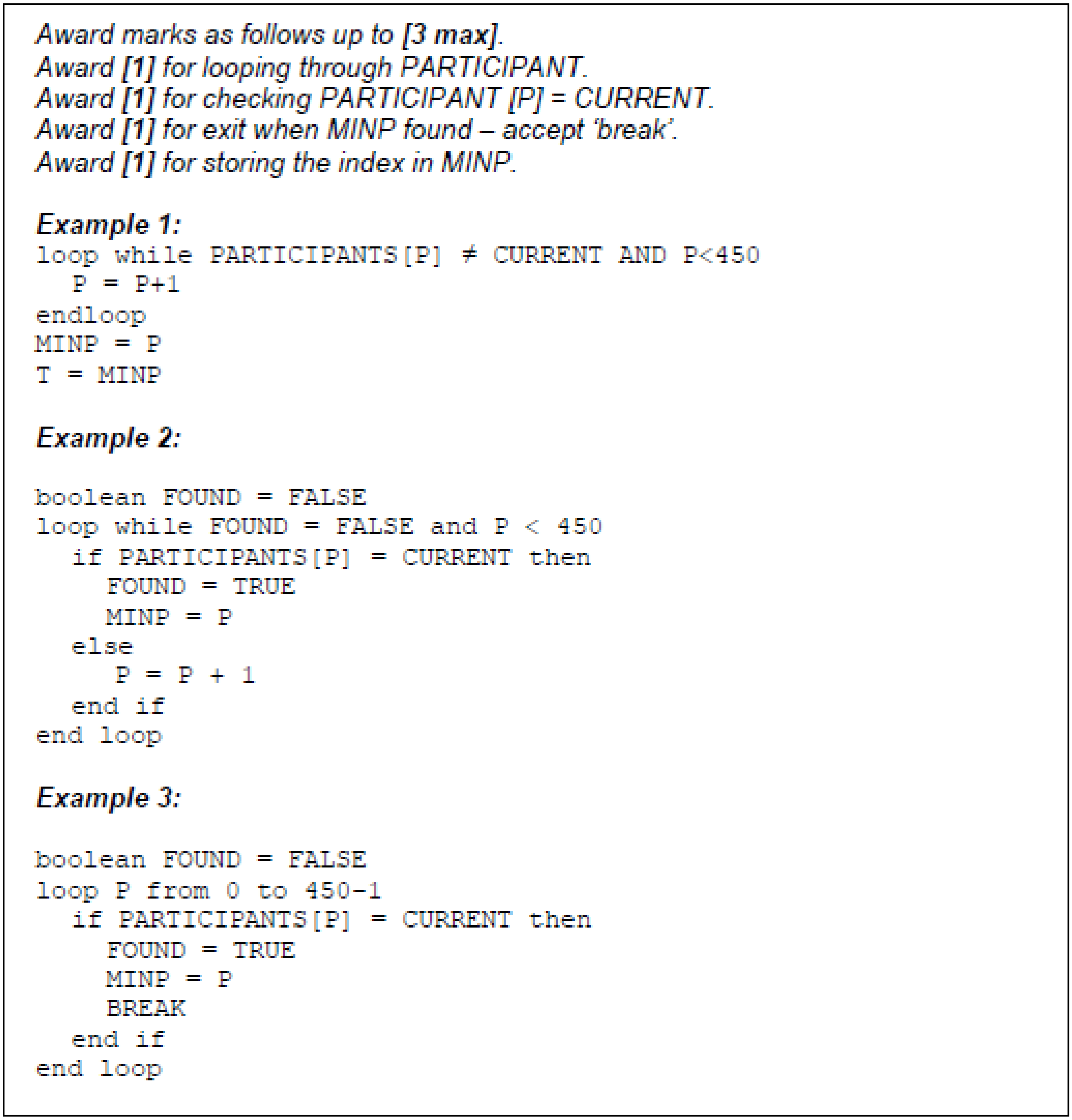
Part of the algorithm that generates the *TEAM* array is shown below, in pseudocode.

```
//Input PARTICIPANTS array, TNAMES collection
TEAM    // array with 450 positions, initialized to '999'
CURRENT // variable to store current name of team;
T, P    // variables to store the indexes of TEAM and PARTICIPANTS,
        // respectively;
MINP    // stores the first index P of members of the CURRENT team;

TNAMES.resetNext()
loop while TNAMES.hasNext()
       CURRENT = TNAME.getNext()
       T = 0; P = 0; MINP = 0     // variables' initialization
       //*
       //* Code to be completed in part (c)(i)
       //*
       //* Code to be completed in part (c)(ii)
       //*
end loop
output TEAM
```

(c) In order to complete this code, and return the correct *TEAM* array,

(i) construct pseudocode to find *MINP*, the first index in *PARTICIPANTS* of the *CURRENT* team, and use it to start the construction of *TEAM* [3]



*Award marks as follows up to* ***[3 max]****.*
*Award [1] for looping through PARTICIPANT.*
*Award [1] for checking PARTICIPANT [P] = CURRENT.*
*Award [1] for exit when MINP found – accept 'break'.*
*Award [1] for storing the index in MINP.*

***Example 1:***

```
loop while PARTICIPANTS[P] ≠ CURRENT AND P<450
  P = P+1
endloop
MINP = P
T = MINP
```

***Example 2:***

```
boolean FOUND = FALSE
loop while FOUND = FALSE and P < 450
  if PARTICIPANTS[P] = CURRENT then
    FOUND = TRUE
    MINP = P
  else
    P = P + 1
  end if
end loop
```

***Example 3:***

```
boolean FOUND = FALSE
loop P from 0 to 450-1
  if PARTICIPANTS[P] = CURRENT then
    FOUND = TRUE
    MINP = P
    BREAK
  end if
end loop
```

(ii) construct pseudocode to find the other participants belonging to the *CURRENT* team, implementing the idea of the closed paths in the *TEAM* array. [4]

*Award marks as follows up to **[4 max]**.*
*Award **[1]** for a correct loop from `P` to 449 (either incremented in (c)(i) or stated here).*
*Award **[1]** for starting at `T=MINP`*
*Award **[1]** for updating `T` index.*
*Award **[1]** for correct value of `P` in `TEAM[T]`.*
*Award **[1]** for closing the path by assigning `MINP`.*

***Example 1:***

```
T = MINP // can be here or in c(i)
loop while P<450
  if PARTICIPANTS[P] = CURRENT then
    TEAM[T] = P
    T = P
  end if
  P = P+1
end loop
TEAM[T] = MINP
```

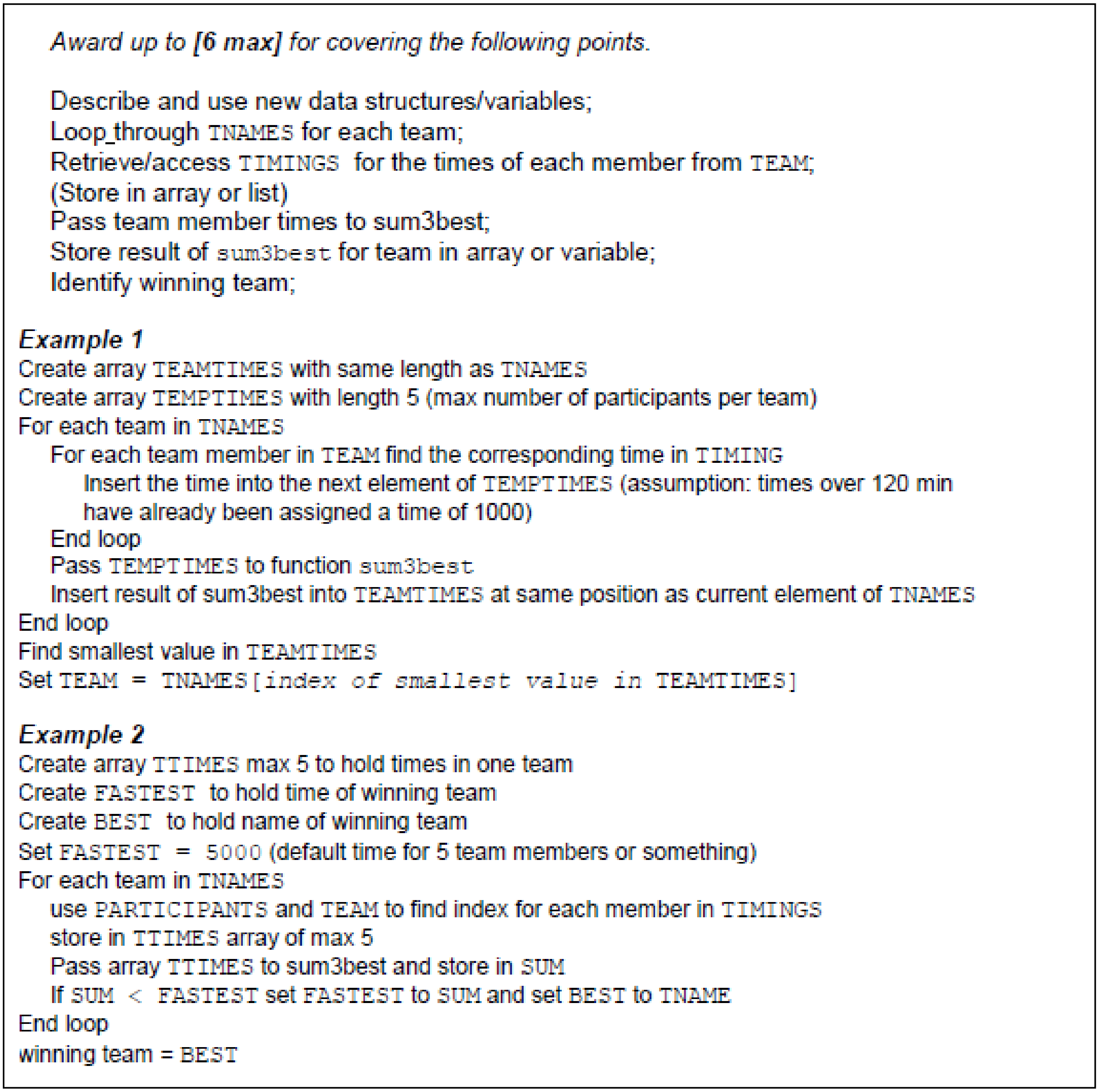
***Example 2:***

```
loop while P<450
  if PARTICIPANTS[P] ≠ CURRENT then
    P = P+1
  else
    TEAM[T] = P
    T = P
    P = P+1
  end if
end loop
TEAM[T] = MINP
```

As part of the program to determine the winning team, an array, *TIMING*, is maintained in parallel to *PARTICIPANTS*. For example, *TIMING[5]* and *PARTICIPANTS[5]* relate to the same participant.

*TIMING* is initialized to zero before the race starts, and updated with the finishing times for each participant. The algorithm *sum3best* is able to output the sum of the three fastest times from any group of times that are passed to the algorithm.

(d) Describe the steps of an algorithm that will find the **winning team**, as defined by the marathon rules on page 6. Clearly mention the use of existing or of new data structures. [6]



*Award up to **[6 max]** for covering the following points.*

Describe and use new data structures/variables;
Loop through TNAMES for each team;
Retrieve/access TIMINGS for the times of each member from TEAM;
(Store in array or list)
Pass team member times to sum3best;
Store result of sum3best for team in array or variable;
Identify winning team;

***Example 1***

```
Create array TEAMTIMES with same length as TNAMES
Create array TEMPTIMES with length 5 (max number of participants per team)
For each team in TNAMES
    For each team member in TEAM find the corresponding time in TIMING
        Insert the time into the next element of TEMPTIMES (assumption: times over 120 min
        have already been assigned a time of 1000)
    End loop
    Pass TEMPTIMES to function sum3best
    Insert result of sum3best into TEAMTIMES at same position as current element of TNAMES
End loop
Find smallest value in TEAMTIMES
Set TEAM = TNAMES[index of smallest value in TEAMTIMES]
```

***Example 2***

```
Create array TTIMES max 5 to hold times in one team
Create FASTEST  to hold time of winning team
Create BEST  to hold name of winning team
Set FASTEST = 5000 (default time for 5 team members or something)
For each team in TNAMES
    use PARTICIPANTS and TEAM to find index for each member in TIMINGS
    store in TTIMES array of max 5
    Pass array TTIMES to sum3best and store in SUM
    If SUM < FASTEST set FASTEST to SUM and set BEST to TNAME
End loop
winning team = BEST
```