[1 Lösungsansatz 2](#_Toc374388960)

[2 Source 4](#_Toc374388961)

[2.1 WorkManagementUnit 4](#_Toc374388962)

[2.2 WorkManagementUnitTest 12](#_Toc374388963)

[3 Tests 18](#_Toc374388964)

[3.1 Reset 18](#_Toc374388965)

[3.2 AddWorkEntry 19](#_Toc374388966)

[3.3 GetTotalWorkTimeForPerson 19](#_Toc374388967)

[3.4 GetAverageWorkTimeForTask 20](#_Toc374388968)

[3.5 PrintWorkSummaryForPerson 21](#_Toc374388969)

[3.6 PrintPersonForTask 21](#_Toc374388970)

[3.7 BusiestPerson 22](#_Toc374388971)

[3.8 GetTotalWorkEntryCount 23](#_Toc374388972)

[3.9 DeletePerson 23](#_Toc374388973)

[3.10 Performance (WorkItem) 24](#_Toc374388974)

[3.11 Performance (Person) 25](#_Toc374388975)

[3.12 Performance (Person + WorkItem) 26](#_Toc374388976)

[4 Diskussion 27](#_Toc374388977)

# Lösungsansatz

In dieser Übung soll das Gedächtnis der WorkManagementUnit durch eine doppelt verkettete zyklische Liste mit Ankerelement ersetzt werden.

Es soll darauf geachtet werden, dass die Schnittstelle sich für den Aufrufer nicht ändert. Da sich die Datenstruktur nun ändert (Person mit Liste der Zeiterfassungen) müssen alle Prozeduren und Funktionen, die mit dem Gedächtnis arbeiten, abgeändert werden, sodass sie mit dem neuen Listentyp und der neuen Datenstruktur umgehen können, ohne dabei ihre Funktion zu ändern.

Obwohl der Typ WorkEntry eigentlich obsolete ist, soll er dennoch beibehalten werden, da es sonst Änderungen an der Schnittstelle geben würde und es für den Aufrufer keinen Unterschied macht, da er vom Aufbau des Gedächtnisses der Unit nicht Bescheid wissen muss. Dieser Typ wir nur beim Einfügen eines neuen Work Entries verwendet, und alle anderen Prozeduren und Funktionen liefern aggregierte Daten und es bestehen keine weiteren Abhängigkeiten zu diesem Datentyp.

Für das Speichern der Einträge soll ein neuer Datentyp eingeführt werden, der die Personen mit einer Liste aller ihrer Arbeitseinträge abbildet. Dieser Datentyp soll in der oben erwähnten doppelt verketteten zyklischen Liste mit Ankerelement gespeichert werden. Die Personen sollen ihrerseits eine einfach verkettete Liste verwenden um die Work Item zu speichern, die zugehörig zur Person sind.

Daher soll auch ein neuer Datentyp spezifiziert werden, welcher die Zeiteinträge für einen Task abbildet.

Da durch diese Änderungen nun keine temporäre Liste in der Funktion BusiestPerson mehr notwendig ist, soll diese Funktion so geändert werden, dass sie mit der dem neuen Gedächtnis arbeitet. Hierbei sollen alle Datentypen, die eigens für diese Funktion implementiert wurden, entfernt werden, sowie auch alle Hilfsfunktionen und Prozeduren. Es soll hier jetzt die Person ausgegeben werden, die die meiste Zeit gearbeitet hat und nicht die Person, die die meisten Einträge hat.

Für das Erstellen der Instanzen, das Einfügen und das Löschen von Personen sollen eigene private Prozeduren implementiert werden, um das Handling mit der Liste zu erleichtern und zu kapseln.

Dieses Mal sollen die Namen und die Bezeichnungen des Tasks mit ignore case behandelt werden und immer mit Kleinbuchstaben gespeichert werden, um Fehler bei den Eingaben zu unterbinden.

Bsp.: Thomas = thOMas oder Implementierung = imPLEmenTIERung. Diese Beispiele zeigen, dass es sich immer um dasselbe Subjekt handelt und lediglich die Groß und Kleinschreibung sie voneinander unterscheidet.

Bei der Funktion DeletePerson soll als Rückgabewert TRUE oder FALSE zurückgegeben werden. TRUE zeigt an, dass eine Person gelöscht wurde und FALSE das die Person nicht gefunden und daher auch nicht gelöscht werden konnte. Beim Löschen einer Person soll darauf geachtet werden, dass die WorkItemList, die von der Person gehalten wird auch vollständig gelöscht wird, bevor die Person gelöscht wird. Auf diesen Aspekt soll auch bei Reset geachtet werden.

Bei der Funktion GetTotalWorkEntryCount soll über alle Einträge aller Personen und ihrer gehaltenen Work Item iteriert werden und die Summe der Iterationen beim Iterieren über die Work Item zurückgegeben werden. Sollten keine Einträge vorhanden sein so soll 0 zurückgegeben werden.

# Source

Folgend sind die Sources der WorkManagementUnit und der WorkManagementUnitTest angeführt. Auf den Source der TimeSpanUnit wurde verzichtet, da es hier keine Änderungen gegeben hat.

## WorkManagementUnit

Folgend ist der Source der WorkManagementUnit angeführt.

{

Unit for handling work entries which are hold by a double linked cyclic list with anchor.

Call Reset to reset the memory.

This unit can handle as much entries as the caller wishes, depends on the

available memory.

}

**UNIT** WorkManagementUnit**;**

{ ######################### Interface part ######################### }

**INTERFACE**

{ Uses the time span unit }

**USES** TImeSpanUnit**;**

{

The types which are visible and usable for the caller.

}

**TYPE**

{ The error codes which can be handled by the caller }

ErrorCode **=** **(**NONE**,** TO\_SHORT**,** TO\_LONG**,** INVALID\_SPAN**);**

{ Compound representing a work entry, which is only used by the caller for adding a entry }

WorkEntry **=** **RECORD**

name**:** **STRING;**

task**:** **STRING;**

spendTime**:** TimeSpan**;**

**END;**

{

Resets the memory of this module by disposing all elements of the list and also disposing all elements

of any contained list of the elements.

}

**PROCEDURE** Reset**;**

{

Creates a work entry for the given data.

@param

name: the name of the person

@param

task: the name of the task the person worked on

@param

spendTime: the TimeSpan instance which represents the time the person has worked

on the defined task

@return

the create workEntry instance

}

**FUNCTION** CreateWorkEntry**(**name**,** task**:** **STRING;** spendTime**:** TimeSpan**):** WorkEntry**;**

{

Adds a WorkEntry to the backed list.

@param

entry: the entry to be added

@param

error: the error parameter which will be set <> NONE if an error occurs

@returns

the occurred error or the given error from the caller which should be NONE

}

**PROCEDURE** AddWorkEntry**(**entry**:** WorkEntry**;** **VAR** error**:** ErrorCode**);**

{

Gets the total work time for the given person.

@param

name: the name of the person

@param

spendTime: the TimeSpan instance given by the caller which will get set with the total work time,

should be initialized by the caller with 0:0:0

@return

the TimeSpan representing the total work time, or all values set to 0 when the person would not be found,

or no entry exists

}

**PROCEDURE** GetTotalWorkTimeForPerson**(**name**:** **STRING;** **VAR** spendTime**:** TimeSpan**);**

{

Gets the average time worked on a task.

@param

task: the task to get the average work time for

@param

average: the TimeSpan instance where the result will be saved, shall be initialized by the

caller with 0:0:0

@return

the TimeSpan instance which contains the average time worked on the task, or all field set to 0,

if the task could not be found, because there can no task be present in the entry holder which has

a time spend less than 1 minute.

}

**PROCEDURE** GetAverageWorkTimeForTask**(**task**:** **STRING;** **VAR** average**:** TimeSpan**);**

{

Prints the persons which has worked on the task, along with their time spend.

@param

task: the task to be printed.

}

**PROCEDURE** PrintPersonForTask**(**task**:** **STRING);**

{

Prints the work summary for the given person.

@param

name: the name of the persons to search for the work summary

}

**PROCEDURE** PrintWorkSummaryForPerson**(**name**:** **STRING);**

{

Gets the name of the person with the most time spent on the saved tasks.

@return

the name of the busiest person

}

**FUNCTION** BusiestPerson**:** **STRING;**

{

Removes all saved entries of a person.

@param

name: the name of the person

@return

true if the person has been removed, false otherwise

}

**FUNCTION** DeletePerson**(**name**:** **STRING):** BOOLEAN**;**

{

Gets the count of all backed work entries.

@return

the count of all work entries

}

**FUNCTION** GetTotalWorkEntryCount**:** LONGINT**;**

{ ######################## Implementation part ######################## }

**IMPLEMENTATION**

**TYPE**

{ Pointer to a WorkItemNode }

WorkItemNode **=** **^**WorkItem**;**

{ Compound representing a work item for the person }

WorkItem **=** **RECORD**

task**:** **STRING;**

spendTime**:** TimeSpan**;**

next**:** WorkItemNode**;**

**END;**

{ Pointer to a list of WorkItemNode }

WorkItemList **=** WorkItemNode**;**

{ Pointer to a PersonItem }

PersonItemNode **=** **^**PersonItem**;**

{ Compound representing the person in the backed list }

PersonItem **=** **RECORD**

name**:** **STRING;**

{ WrokItemList with all work items related to this person }

workItems**:** WorkItemList**;**

prev**,** next**:** PersonItemNode**;**

**END;**

{ Pointer to the list of PersonItem }

PersonItemList **=** PersonItemNode**;**

{

The memory which contains the persons saved in the system.

Implemented as a double linked cyclic list with anchor

}

**VAR**

list**:** PersonItemList**;**

{ ######################### Private Function/Procedure part ######################### }

{

Creates a WorkItem instance.

@param

entry the WorkEntry instance to create the WorkItem instance for

@return

the created WorkItem instance

}

**FUNCTION** CreateWorkItemNode**(**entry**:** WorkEntry**):** WorkItemNode**;**

**VAR**

node**:** WorkItemNode**;**

**BEGIN**

New**(**node**);**

node**^.**task **:=** LowerCase**(**entry**.**task**);**

node**^.**spendTime **:=** entry**.**spendTime**;**

node**^.**next **:=** **NIL;**

CreateWorkItemNode **:=** node**;**

**END;**

{

Creates a PersonItem instance.

@param

entry: the entry to create PersonItem instance for

@return

the created PersonItem instance

}

**FUNCTION** CreatePersonItemNode**(**entry**:** WorkEntry**):** PersonItemNode**;**

**VAR**

node**:** PersonItemNode**;**

**BEGIN**

New**(**node**);**

node**^.**name **:=** LowerCase**(**entry**.**name**);**

node**^.**workItems **:=** CreateWorkItemNode**(**entry**);**

node**^.**prev **:=** node**;**

node**^.**next **:=** node**;**

CreatePersonItemNode **:=** node**;**

**END;**

{

Adds a PersonItem instance to the backed list.

@param

entry the WorkEntry instance to create a PersonItem instance for which gets added to the backed list

}

**PROCEDURE** AppendPersonNode**(**entry**:** WorkEntry**);**

**VAR**

node**:** PersonItemNode**;**

**BEGIN**

node **:=** CreatePersonItemNode**(**entry**);**

node**^.**next **:=** list**;**

node**^.**prev **:=** list**^.**prev**;**

list**^.**prev**^.**next **:=** node**;**

list**^.**prev **:=** node**;**

**END;**

{

Adds a WorkItemNode instance to the beginning of the WrokItemList of the PersonItem.

It is assumed that the given PeronItem contains at least one WorkItem which is guaranteed by the implementation.

@param

person: the person to add the WorkItemNode instance to its backed workItem list

@param

entry: the entry which contains the information for the WorkItemNode instance

}

**PROCEDURE** PrependWorkItemNode**(**person**:** PersonItemNode**;** entry**:** WorkEntry**);**

**VAR**

node**:** WorkItemNode**;**

**BEGIN**

node **:=** CreateWorkItemNode**(**entry**);**

node**^.**next **:=** person**^.**workItems**;**

person**^.**workItems **:=** node**;**

**END;**

{

Disposes all WorkItemNode of the given WorkItemList instance.

@param

workItemList: the list of WorkItemNode instances

@return

the disposed WorkitemList which is being to be NIL

}

**PROCEDURE** DisposeWorkItemList**(VAR** workItemList**:** WorkItemList**);**

**VAR**

next**:** WorkItemNode**;**

**BEGIN**

(\* { Test output begin }

IF (workItemList = NIL) THEN BEGIN

WriteLn('No WorkItems to dispose');

END;

{ Test output end } \*)

**WHILE** **(**workItemList **<>** **NIL)** **DO** **BEGIN**

next **:=** workItemList**^.**next**;**

(\* WriteLn('Dispose WorkItem: ', workItemList^.task:15, ' | spendTime: ', TimeSpanUnit.TimeSpanToString(workItemList^.spendTime)); \*)

Dispose**(**workItemList**);**

workItemList **:=** next**;**

**END;**

**END;**

{

Finds a person by its name.

@param

name: the name of the person which will be handled with lower case

@return

the PersonItemNode if the person could be found NIL otherwise

}

**FUNCTION** FindPersonItemNodeForName**(**name**:** **STRING):** PersonItemNode**;**

**VAR**

node**:** PersonItemNode**;**

lowerName**:** **STRING;**

**BEGIN**

lowerName **:=** LowerCase**(**name**);**

FindPersonItemNodeForName **:=** **NIL;**

node **:=** list**^.**next**;**

**WHILE** **((**node **<>** list**)** **AND** **(**node**^.**name **<>** lowerName**))** **DO** **BEGIN**

node **:=** node**^.**next**;**

**END;**

{ Check if name has been found and exclude anchor }

**IF** **(**node **<>** list**)** **THEN** **BEGIN**

FindPersonItemNodeForName **:=** node**;**

**END;**

**END;**

{

Finds the WorkItems of the given PersonItemNode instance for the given task.

ATTENTION: The returned list needs to be disposed

@param

person: the PersonItemNode instance to search for the given task on its WorkItemList

@param

task: the task to be searched

@return

the created WorkItemList instance if at least one entry for the task could be found, NIL otherwise

}

**FUNCTION** FindWorkItemNodesForTask**(**person**:** PersonItemNode**;** task**:** **STRING):** WorkItemList**;**

**VAR**

node**,** newNode**:** WorkItemNode**;**

itemList**:** WorkItemList**;**

lowerTask**:** **STRING;**

**BEGIN**

itemList **:=** **NIL;**

lowerTask **:=** LowerCase**(**task**);**

node **:=** person**^.**workItems**;**

**WHILE** **(**node **<>** **NIL)** **DO** **BEGIN**

**IF** **(**node**^.**task **=** lowerTask**)** **THEN** **BEGIN**

{ Clone WorkItem node and put into temp list}

New**(**newNode**);**

newNode**^.**task **:=** node**^.**task**;**

newNode**^.**spendTime **:=** node**^.**spendTime**;**

newNode**^.**next **:=** **NIL;**

**IF** **(**itemList **=** **NIL)** **THEN** **BEGIN**

itemList **:=** newNode**;**

**END**

**ELSE** **BEGIN**

newNode**^.**next **:=** itemList**;**

itemList **:=** newNode**;**

**END;**

**END;**

node **:=** node**^.**next**;**

**END;**

FindWorkItemNodesForTask **:=** itemList**;**

**END;**

{

Answers the question if the given PersonItemNode instance contains at leat one WrokItemNode instance within its workItems

for the given task.

@param

person: the PersonItemNode instance to search for the given task on its workItems

@param

task: the task to search for

@retrun

true if the PersonItemNode instance contains at least one WorkItemNode instance with the given task defined, false otherwise

}

**FUNCTION** PersonHasTask**(**person**:** PersonItemNode**;** task**:** **STRING):** BOOLEAN**;**

**VAR**

node**:** WorkItemNode**;**

lowerTask**:** **STRING;**

**BEGIN**

lowerTask **:=** LowerCase**(**task**);**

node **:=** person**^.**workItems**;**

**WHILE** **((**node **<>** **NIL)** **AND** **(**node**^.**task **<>** lowerTask**))** **DO** **BEGIN**

node **:=** node**^.**next**;**

**END;**

PersonHasTask **:=** **(**node **<>** **NIL);**

**END;**

{

Gets the sum of all TimeSpan instances hold by the WorkItemNodes and also the count of WorkItem in the list

@param

itemList: the list of WorkItemNode to sum their backed TimeSpan instances

@param

the seconds to build the summary

@param

the count to increase by one for each WorkItemNode

@return

the sum of all TimeSpan instances and the count of entries of the given WorkItemList.

}

**PROCEDURE** SumSpendTimeOfWorkItemList**(**itemList**:** WorkItemList**;** **VAR** seconds**:** LONGINT**;** **VAR** count**:** LONGINT**);**

**BEGIN**

**WHILE** **(**itemList **<>** **NIL)** **DO** **BEGIN**

seconds **:=** seconds **+** TimeSpanUnit**.**TimeSpanToSeconds**(**itemList**^.**spendTime**).**timeInSeconds**;**

Inc**(**count**);**

itemList **:=** itemList**^.**next**;**

**END;**

**END;**

{ ######################### Public Function/Procedure part ######################### }

{ Reset memory }

**PROCEDURE** Reset**;**

**VAR**

node**,** next**:** PersonItemNode**;**

**BEGIN**

node **:=** list**^.**next**;**

(\* { Test output begin }

WriteLn;

WriteLn('---------------------');

WriteLn('Disposing all Persons');

WriteLn('---------------------');

IF (node = list) THEN BEGIN

WriteLn('No Persons to dispose');

WriteLn;

END;

{ Test output end } \*)

**WHILE** **(**node **<>** list**)** **DO** **BEGIN**

next **:=** node**^.**next**;**

(\* WriteLn('Dispose WorkItems for: ', node^.name:10); \*)

DisposeWorkItemList**(**node**^.**workItems**);**

(\* WriteLn('Dispose Person: ', node^.name:10);

WriteLn;\*)

Dispose**(**node**);**

node **:=** next**;**

**END;**

list**^.**prev **:=** list**;**

list**^.**next **:=** list**;**

**END;**

{ Creates a WorkEntry instance }

**FUNCTION** CreateWorkEntry**(**name**,** task**:** **STRING;** spendTime**:** TimeSpan**):** WorkEntry**;**

**BEGIN**

CreateWorkEntry**.**name **:=** LowerCase**(**name**);**

CreateWorkEntry**.**task **:=** LowerCase**(**task**);**

CreateWorkEntry**.**spendTime **:=** spendTime**;**

**END;**

{ Adds the work entry to the memory}

**PROCEDURE** AddWorkEntry**(**entry**:** WorkEntry**;** **VAR** error**:** ErrorCode**);**

**VAR**

person**:** PersonItemNode**;**

**BEGIN**

{ Converts the time span to seconds, and will contain any error }

entry**.**spendTime **:=** TimeSpanToSeconds**(**entry**.**spendTime**);**

{ Invalid time span detected }

**IF** **(**entry**.**spendTime**.**error **<>** ''**)** **THEN** **BEGIN**

error **:=** INVALID\_SPAN

**END**

{ To less time span }

**ELSE** **IF** **(**entry**.**spendTime**.**timeInSeconds **<** 60**)** **THEN** **BEGIN**

error **:=** TO\_SHORT

**END**

{ To much time }

**ELSE** **IF** **(**entry**.**spendTime**.**timeInSeconds **>** **(**8 **\*** 60 **\*** 60**))** **THEN** **BEGIN**

error **:=** TO\_LONG

**END**

{ valid to add entry }

**ELSE** **BEGIN**

person **:=** FindPersonItemNodeForName**(**entry**.**name**);**

{ If a entry already exists for the given name }

**IF** **(**person **<>** **NIL)** **THEN** **BEGIN**

PrependWorkItemNode**(**person**,** entry**);**

**END**

{ If new person has to be added }

**ELSE** **BEGIN**

AppendPersonNode**(**entry**);**

**END;**

**END;**

**END;**

{ Gets total work time }

**PROCEDURE** GetTotalWorkTimeForPerson**(**name**:** **STRING;** **VAR** spendTime**:** TimeSpan**);**

**VAR**

node**:** PersonItemNode**;**

sec**,** count**:** LONGINT**;**

**BEGIN**

sec **:=** 0**;**

count **:=** 0**;**

node **:=** FindPersonItemNodeForName**(**name**);**

**IF** **(**node **<>** **NIL)** **THEN** **BEGIN**

SumSpendTimeOfWorkItemList**(**node**^.**workItems**,** sec**,** count**);**

**END;**

spendTime **:=** TimeSpanUnit**.**SecondsToTimeSpan**(**sec**);**

**END;**

{ Gets average of spend time }

**PROCEDURE** GetAverageWorkTimeForTask**(**task**:** **STRING;** **VAR** average**:** TimeSpan**);**

**VAR**

node**:** PersonItemNode**;**

items**:** WorkItemList**;**

sec**,** count**:** LONGINT**;**

**BEGIN**

sec **:=** 0**;**

count **:=** 0**;**

average**.**timeInSeconds **:=** 0**;**

{ Only get average time when entries are present }

node **:=** list**^.**next**;**

**WHILE** **(**node **<>** list**)** **DO** **BEGIN**

{ Search for this task }

items **:=** FindWorkItemNodesForTask**(**node**,** task**);**

{ Build sum over the Work Items }

SumSpendTimeOfWorkItemList**(**items**,** sec**,** count**);**

{ Dispose WorkItem List because it has been copied as interface spec says }

DisposeWorkItemList**(**items**);**

node **:=** node**^.**next**;**

**END;**

{ Calculate the average if the task on working persons have been found }

**IF** **(**sec **<>** 0**)** **THEN** **BEGIN**

sec **:=** **(**sec **DIV** count**);**

**END;**

{ Create time span out of seconds }

average **:=** TimeSpanUnit**.**SecondsToTimeSpan**(**sec**);**

**END;**

{ Print persons for task to table }

**PROCEDURE** PrintPersonForTask**(**task**:** **STRING);**

**VAR**

node**:** PersonItemNode**;**

count**:** INTEGER**;**

**BEGIN**

count **:=** 0**;**

WriteLn**(**'################### Persons working on task ###################'**);**

WriteLn**(**'Task : '**,** task**);**

WriteLn**(**'Persons : '**);**

{ Only print result if there are entries available }

node **:=** list**^.**next**;**

**WHILE** **(**node **<>** list**)** **DO** **BEGIN**

{ Search for this task }

**IF** **(**PersonHasTask**(**node**,** task**))** **THEN** **BEGIN**

Inc**(**count**);**

WriteLn**(**' '**,** count**,** ': '**,** node**^.**name**);**

**END;**

node **:=** node**^.**next**;**

**END;**

{ Either no entries or task not found }

**IF** **(**count **=** 0**)** **THEN** **BEGIN**

WriteLn**(**' No work entries found for the task !!!'**);**

**END;**

WriteLn**(**'################### Persons working on task ###################'**);**

**END;**

{ Print work summary for a person }

**PROCEDURE** PrintWorkSummaryForPerson**(**name**:** **STRING);**

**VAR**

person**:** PersonItemNode**;**

node**:** WorkItemNode**;**

**BEGIN**

WriteLn**(**'################### Work summary for person ###################'**);**

WriteLn**(**'Person : '**,** name**);**

{ Only print result if a entry for the person exists }

person **:=** FindPersonItemNodeForName**(**name**);**

**IF** **(**person **<>** **NIL)** **THEN** **BEGIN**

node **:=** person**^.**workItems**;**

**WHILE** **(**node **<>** **NIL)** **DO** **BEGIN**

WriteLn**(**' Task: '**,** node**^.**task**:**15**,** ' | Spend time: '**,** TimeSpanUnit**.**TimeSpanToString**(**node**^.**spendTime**));**

node **:=** node**^.**next**;**

**END;**

**END**

{ Either no entries or person not found }

**ELSE** **BEGIN**

WriteLn**(**' No work entries found for the person !!!'**:**10**);**

**END;**

WriteLn**(**'################### Work summary for person ###################'**);**

**END;**

{ Get busiest person }

**FUNCTION** BusiestPerson**:** **STRING;**

**VAR**

node**:** PersonItemNode**;**

oldSeconds**,** seconds**,** count**:** LONGINT**;**

**BEGIN**

oldSeconds **:=** 0**;**

count **:=** 0**;**

BusiestPerson **:=** ''**;**

node **:=** list**^.**next**;**

{ iterate over all work entries }

**WHILE** **(**node **<>** list**)** **DO** **BEGIN**

seconds **:=** 0**;**

SumSpendTimeOfWorkItemList**(**node**^.**workItems**,** seconds**,** count**);**

**if** **(**seconds **>** oldSeconds**)** **THEN** **BEGIN**

oldSeconds **:=** seconds**;**

BusiestPerson **:=** node**^.**name**;**

**END;**

node **:=** node**^.**next**;**

**END;**

**END;**

{ Removes a person }

**FUNCTION** DeletePerson**(**name**:** **STRING):** BOOLEAN**;**

**VAR**

person**:** PersonItemNode**;**

**BEGIN**

DeletePerson **:=** false**;**

person **:=** FindPersonItemNodeForName**(**name**);**

**IF** **(**person **<>** **NIL)** **THEN** **BEGIN**

person**^.**prev**^.**next **:=** person**^.**next**;**

person**^.**next**^.**prev **:=** person**^.**prev**;**

DisposeWorkItemList**(**person**^.**workItems**);**

(\* WriteLn('Dispose Person: ', person^.name); \*)

Dispose**(**person**);**

DeletePerson **:=** true**;**

**END;**

**END;**

{ Get work entry count }

**FUNCTION** GetTotalWorkEntryCount**:** LONGINT**;**

**VAR**

personNode**:** PersonItemNode**;**

workNode**:** WorkItemNode**;**

count**:** LONGINT**;**

**BEGIN**

count **:=** 0**;**

personNode **:=** list**^.**next**;**

**WHILE** **(**personNode **<>** list**)** **DO** **BEGIN**

workNode **:=** personNode**^.**workItems**;**

**WHILE** **(**workNode **<>** **NIL)** **DO** **BEGIN**

Inc**(**count**);**

workNode **:=** workNode**^.**next**;**

**END;**

personNode **:=** personNode**^.**next**;**

**END;**

GetTotalWorkEntryCount **:=** count**;**

**END;**

**BEGIN**

{ Initializes the list by creating the anchor for the list which is supposed to be never deleted }

list **:=** CreatePersonItemNode**(**CreateWorkEntry**(**'Not meant to be used'**,** 'Not meant to be used'**,** TimeSpanUnit**.**CreateTimeSpan**(-**1**,** **-**1**,** **-**1**)));**

**END.**

## WorkManagementUnitTest

Folgend ist der Source der WorkManagementUnitTest angeführt.

**PROGRAM** WorkManagementUnitTest**;**

**USES** WorkManagementUnit**,** TimeSpanUnit**;**

{

Adds a entry and prints the added entry to the console

}

**PROCEDURE** AddEntry**(**name**,** task**:** **STRING;** span**:** TimeSpan**);**

**VAR**

entry**:** WorkEntry**;**

error**:** ErrorCode**;**

**BEGIN**

error **:=** NONE**;**

entry **:=** WorkManagementUnit**.**CreateWorkEntry**(**name**,** task**,** span**);**

WorkManagementUnit**.**AddWorkEntry**(**entry**,** error**);**

WriteLn**(**'Error: '**,** error**:**13**,** 'Name: '**,** entry**.**name**:**10**,** ' | Task: '**,** entry**.**task**:**15**,** ' | Time: '**,** TimeSpanUnit**.**TimeSpanToString**(**entry**.**spendtime**));**

**END;**

{

Tests GetTotalWorkTimeForPerson

}

**PROCEDURE** TestGetTotalWorkTimeForPerson**;**

**VAR**

span**:** TimeSpan**;**

**BEGIN**

WriteLn**(**'---------------------- TestGetTotalWorkTimeForPerson ----------------------'**);**

WorkManagementUnit**.**Reset**();**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 0**);**

WriteLn**(**'No entries are present: '**);**

WorkManagementUnit**.**GetTotalWorkTimeForPerson**(**'Thomas'**,** span**);**

WriteLn**(**'Total work time for Thomas: '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

WriteLn**;**

WorkManagementUnit**.**Reset**();**

AddEntry**(**'Thomas' **,** 'Impl'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

AddEntry**(**'Thomas' **,** 'Spec'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

AddEntry**(**'Thomas' **,** 'Test'**,** TimeSpanUnit**.**CreateTimeSpan**(**1**,** 0**,** 0**));**

WorkManagementUnit**.**GetTotalWorkTimeForPerson**(**'Maria'**,** span**);**

WriteLn**(**'Total work time for Maria: '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

WriteLn**;**

WorkManagementUnit**.**Reset**();**

AddEntry**(**'Thomas' **,** 'Impl'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

AddEntry**(**'Thomas' **,** 'Spec'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

AddEntry**(**'Thomas' **,** 'Test'**,** TimeSpanUnit**.**CreateTimeSpan**(**1**,** 0**,** 0**));**

AddEntry**(**'Hannes' **,** 'Impl'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

AddEntry**(**'Hannes' **,** 'Spec'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

AddEntry**(**'Hannes' **,** 'Test'**,** TimeSpanUnit**.**CreateTimeSpan**(**2**,** 2**,** 0**));**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 0**);**

WorkManagementUnit**.**GetTotalWorkTimeForPerson**(**'Thomas'**,** span**);**

WriteLn**;**

WriteLn**(**'Total work time for Thomas: '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 0**);**

WorkManagementUnit**.**GetTotalWorkTimeForPerson**(**'Hannes'**,** span**);**

WriteLn**(**'Total work time for Hannes: '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

WorkManagementUnit**.**Reset**();**

WriteLn**(**'---------------------- TestGetTotalWorkTimeForPerson ----------------------'**);**

WriteLn**;**

**END;**

{

Tests the AddWorkEntry

}

**PROCEDURE** TestAddWorkEntry**;**

**BEGIN**

WriteLn**(**'---------------------- TestAddWorkEntry ----------------------'**);**

WorkManagementUnit**.**Reset**();**

AddEntry**(**'Thomas'**,** 'Impl'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 1**));**

AddEntry**(**'THomas'**,** 'IMPL'**,** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 1**));**

AddEntry**(**'THomas'**,** 'IMPL'**,** TimeSpanUnit**.**CreateTimeSpan**(**1**,** 60**,** 60**));**

AddEntry**(**'HaNNes' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

AddEntry**(**'HaNes' **,** 'SpecificaTIon'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

AddEntry**(**'Maria' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**));**

AddEntry**(**'MAria' **,** 'SpecifICation'**,** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**));**

WorkManagementUnit**.**Reset**();**

WriteLn**(**'---------------------- TestAddWorkEntry ----------------------'**);**

WriteLn**;**

**END;**

{

Tests the procedure Rest

}

**PROCEDURE** TestReset**;**

**BEGIN**

WriteLn**(**'---------------------- TestReset ----------------------'**);**

WorkManagementUnit**.**Reset**();**

AddEntry**(**'Thomas'**,** 'Impl'**,** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**));**

AddEntry**(**'Thomas'**,** 'Impl'**,** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**));**

AddEntry**(**'Thomas'**,** 'Spec'**,** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**));**

AddEntry**(**'Thomas'**,** 'Spec'**,** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**));**

AddEntry**(**'Thomas'**,** 'Test'**,** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**));**

AddEntry**(**'Thomas'**,** 'Test'**,** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**));**

AddEntry**(**'Hannes' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**));**

AddEntry**(**'Hannes' **,** 'Testing'**,** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**));**

AddEntry**(**'Hannes' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**));**

WorkManagementUnit**.**Reset**();**

WriteLn**(**'---------------------- TestReset ----------------------'**);**

WriteLn**;**

**END;**

{

Tests the GetAverageWorkTimeForTask

}

**PROCEDURE** TestGetAverageWorkTimeForTask**;**

**VAR**

span**:** TImeSpan**;**

**BEGIN**

WriteLn**(**'---------------------- TestGetAverageWorkTimeForTask ----------------------'**);**

WorkManagementUnit**.**Reset**();**

WriteLn**(**'No entries are present: '**);**

WorkManagementUnit**.**GetAverageWorkTimeForTask**(**'Doku'**,** span**);**

WriteLn**(**'AverageTime for Doku: '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

WriteLn**;**

AddEntry**(**'Thomas' **,** 'Impl'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

AddEntry**(**'Thomas' **,** 'Impl'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

WorkManagementUnit**.**GetAverageWorkTimeForTask**(**'Doku'**,** span**);**

WriteLn**(**'AverageTime for Doku: '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

WriteLn**;**

WorkManagementUnit**.**Reset**();**

AddEntry**(**'Thomas' **,** 'Impl'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

AddEntry**(**'Thomas' **,** 'Impl'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

AddEntry**(**'Thomas' **,** 'Spec'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

AddEntry**(**'Thomas' **,** 'Spec'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

AddEntry**(**'Hannes' **,** 'Spec'**,** TimeSpanUnit**.**CreateTimeSpan**(**1**,** 0**,** 0**));**

AddEntry**(**'Hannes' **,** 'Spec'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

AddEntry**(**'Hannes' **,** 'Impl'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

AddEntry**(**'Hannes' **,** 'Impl'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

WriteLn**;**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 0**);**

WorkManagementUnit**.**GetAverageWorkTimeForTask**(**'Impl'**,** span**);**

WriteLn**;**

WriteLn**(**'Average time for Impl: '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

span **:=** TimeSpanUnit**.**createTimeSpan**(**0**,** 0**,** 0**);**

WorkManagementUnit**.**GetAverageWorkTimeForTask**(**'Spec'**,** span**);**

WriteLn**;**

WriteLn**(**'Average time for Spec: '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

WorkManagementUnit**.**Reset**();**

WriteLn**(**'---------------------- TestGetAverageWorkTimeForTask ----------------------'**);**

WriteLn**;**

**END;**

{

Tests the procedure PrintPersonForTask

}

**PROCEDURE** TestPrintPersonForTask**;**

**BEGIN**

WorkManagementUnit**.**Reset**();**

WriteLn**(**'No entries are present: '**);**

WorkManagementUnit**.**PrintPersonForTask**(**'Implementation'**);**

WorkManagementUnit**.**Reset**();**

AddEntry**(**'Maria' **,** 'Test'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

AddEntry**(**'Thomas' **,** 'Test'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

WorkManagementUnit**.**PrintPersonForTask**(**'Implementation'**);**

WriteLn**;**

WriteLn**(**'---------------------- TestPrintPersonForTask ----------------------'**);**

AddEntry**(**'Maria' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

AddEntry**(**'Thomas' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

AddEntry**(**'Maria' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**1**,** 0**,** 0**));**

AddEntry**(**'Markus' **,** 'Test'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

AddEntry**(**'Hannes' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

AddEntry**(**'Hannes' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

AddEntry**(**'Hannes' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**1**,** 1**,** 1**));**

AddEntry**(**'Hannes' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

AddEntry**(**'Julia' **,** 'Doku'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

WriteLn**;**

WorkManagementUnit**.**PrintPersonForTask**(**'Implementation'**);**

WorkManagementUnit**.**Reset**();**

WriteLn**(**'---------------------- TestPrintPersonForTask ----------------------'**);**

WriteLn**;**

**END;**

{

Test the procedure PrintWorkSummaryForPerson.

}

**PROCEDURE** TestPrintWorkSummaryForPerson**;**

**BEGIN**

WorkManagementUnit**.**Reset**();**

WriteLn**(**'---------------------- TestPrintWorkSummaryForPerson ----------------------'**);**

WriteLn**(**'No entries are present'**);**

WorkManagementUnit**.**PrintWorkSummaryForPerson**(**'Thomas'**);**

WriteLn**;**

AddEntry**(**'Hannes' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**1**,** 1**,** 1**));**

AddEntry**(**'Hannes' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

WorkManagementUnit**.**PrintWorkSummaryForPerson**(**'Thomas'**);**

WriteLn**;**

AddEntry**(**'Thomas' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 2**));**

AddEntry**(**'Thomas' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

AddEntry**(**'Thomas' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**1**,** 1**,** 1**));**

AddEntry**(**'Thomas' **,** 'Test'**,** TimeSpanUnit**.**CreateTimeSpan**(**2**,** 2**,** 2**));**

AddEntry**(**'Hannes' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**));**

AddEntry**(**'Hannes' **,** 'Test'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 2**));**

AddEntry**(**'Hannes' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**1**,** 1**,** 1**));**

AddEntry**(**'Hannes' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

WorkManagementUnit**.**PrintWorkSummaryForPerson**(**'Thomas'**);**

WriteLn**;**

WorkManagementUnit**.**PrintWorkSummaryForPerson**(**'Hannes'**);**

WorkManagementUnit**.**Reset**();**

WriteLn**(**'---------------------- TestPrintWorkSummaryForPerson ----------------------'**);**

WriteLn**;**

**END;**

{

Tests the function BusiestPerson

}

**PROCEDURE** TestBusiestPerson**;**

**VAR**

name**:** **STRING;**

**BEGIN**

WriteLn**(**'---------------------- TestBusiestPerson ----------------------'**);**

WorkManagementUnit**.**Reset**();**

WriteLn**(**'No entries:'**);**

WriteLn**(**'Buisiest person: '**,** WorkManagementUnit**.**BusiestPerson**);**

WriteLn**;**

WriteLn**(**'single result:'**);**

AddEntry**(**'Thomas' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

AddEntry**(**'Thomas' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 2**));**

AddEntry**(**'Maria' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

AddEntry**(**'Maria' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

AddEntry**(**'Hannes' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

AddEntry**(**'Hannes' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

name **:=** WorkManagementUnit**.**BusiestPerson**;**

WriteLn**(**'Buisiest person: '**,** name**);**

WriteLn**;**

WriteLn**(**'multiple result:'**);**

AddEntry**(**'Thomas' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

AddEntry**(**'Thomas' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 2**));**

AddEntry**(**'Maria' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 2**));**

AddEntry**(**'Maria' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

AddEntry**(**'Hannes' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

AddEntry**(**'Hannes' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**));**

name **:=** WorkManagementUnit**.**BusiestPerson**;**

WriteLn**(**'Buisiest person: '**,** name**);**

WorkManagementUnit**.**Reset**();**

WriteLn**(**'---------------------- TestBusiestPerson ----------------------'**);**

WriteLn**;**

**END;**

{

Tests the function GetToalWorkEntryCount

}

**PROCEDURE** TestGetToalWorkEntryCount**;**

**VAR**

i**,** count**:** LONGINT**;**

it**:** **STRING;**

**BEGIN**

count **:=** 10**;**

WriteLn**(**'---------------------- TestGetToalWorkEntryCount ----------------------'**);**

WorkManagementUnit**.**Reset**();**

WriteLn**(**'No entries:'**);**

WriteLn**(**'Total work entry count: '**,** WorkManagementUnit**.**GetTotalWorkEntryCount**);**

WriteLn**;**

WriteLn**(**'Adding '**,** count**,** ' different persons with each two work entry'**);**

**FOR** i **:=** 1 **TO** count **DO** **BEGIN**

Str**(**i**,** it**);**

AddEntry**((**it **+** '-Thomas'**)** **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

AddEntry**((**it **+** '-Thomas'**)** **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

**END;**

WriteLn**(**'Total work entry count: '**,** WorkManagementUnit**.**GetTotalWorkEntryCount**);**

WriteLn**(**'---------------------- TestGetToalWorkEntryCount ----------------------'**);**

**END;**

{

Tests the function DeletePerson

}

**PROCEDURE** TestDeletePerson**;**

**VAR**

result**:** BOOLEAN**;**

**BEGIN**

WriteLn**(**'---------------------- TestGetToalWorkEntryCount ----------------------'**);**

WorkManagementUnit**.**Reset**();**

WriteLn**(**'No entries:'**);**

WriteLn**(**'Deleted: '**,** WorkManagementUnit**.**DeletePerson**(**'thomas'**));**

WriteLn**;**

WorkManagementUnit**.**Reset**();**

AddEntry**(**'Thomas' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

AddEntry**(**'Thomas' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 2**));**

AddEntry**(**'Maria' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 2**));**

AddEntry**(**'Maria' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

WriteLn**;**

WriteLn**(**'Deleted: '**,** WorkManagementUnit**.**DeletePerson**(**'hannes'**));**

WriteLn**;**

WorkManagementUnit**.**Reset**();**

AddEntry**(**'Hannes' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

AddEntry**(**'Hannes' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 2**));**

AddEntry**(**'Thomas' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

AddEntry**(**'Thomas' **,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 2**));**

AddEntry**(**'Maria' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 2**));**

AddEntry**(**'Maria' **,** 'Specification'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

WriteLn**;**

result **:=** WorkManagementUnit**.**DeletePerson**(**'thomas'**);**

WriteLn**;**

WriteLn**(**'Deleted: '**,** result**);**

WriteLn**;**

WriteLn**(**'Total work entry count: '**,** WorkManagementUnit**.**GetTotalWorkEntryCount**);**

WriteLn**;**

WorkManagementUnit**.**PrintWorkSummaryForPerson**(**'maria'**);**

WriteLn**;**

WorkManagementUnit**.**PrintWorkSummaryForPerson**(**'hannes'**);**

WriteLn**(**'---------------------- TestGetToalWorkEntryCount ----------------------'**);**

**END;**

{

Tests the performance when work items are saved for one person

}

**PROCEDURE** TestPerformanceWorkEntry**;**

**VAR**

i**:** LONGINT**;**

it**:** **STRING;**

**BEGIN**

**FOR** i **:=** 1 **TO** 20000 **DO** **BEGIN**

Str**(**i**,** it**);**

Write**(**it**,** ':'**);**

AddEntry**(**'-Thomas'**,** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

**END;**

WriteLn**(**'Finished Insert'**);**

WriteLn**(**'Total work entry count: '**,** WorkManagementUnit**.**GetTotalWorkEntryCount**);**

**END;**

{

Tests the performance when multiple persons are saved with each one work item

}

**PROCEDURE** TestPerformancePersons**;**

**VAR**

i**:** LONGINT**;**

it**:** **STRING;**

**BEGIN**

**FOR** i **:=** 1 **TO** 20000 **DO** **BEGIN**

Str**(**i**,** it**);**

AddEntry**((**it **+** '-Thomas'**),** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

**END;**

WriteLn**(**'Finished Insert'**);**

WriteLn**(**'Total work entry count: '**,** WorkManagementUnit**.**GetTotalWorkEntryCount**);**

**END;**

{

Tests the performance when multiple persons are saved with each 10 additonal work items

}

**PROCEDURE** TestPerformancePersonWorkEntry**;**

**VAR**

i**,** j**:** LONGINT**;**

it**,** it2**:** **STRING;**

**BEGIN**

**FOR** i **:=** 1 **TO** 20000 **DO** **BEGIN**

Str**(**i**,** it**);**

AddEntry**((**it **+** '-Thomas'**),** 'Implementation'**,** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

WriteLn**(**'Adding 10 additional work items: '**);**

**FOR** j **:=** 1 **TO** 10 **DO** **BEGIN**

Str**(**j**,** it2**);**

AddEntry**((**it **+** '-Thomas'**),** **(**it2 **+** '-Implementation'**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 1**));**

**END;**

**END;**

WriteLn**(**'Finished Insert'**);**

WriteLn**(**'Total work entry count: '**,** WorkManagementUnit**.**GetTotalWorkEntryCount**);**

**END;**

**BEGIN**

{ Test TestReset }

TestReset**;**

{ Test AddWorkEntry }

TestAddWorkEntry**;**

{ TestGetTotalWorkTimeForPerson }

TestGetTotalWorkTimeForPerson**;**

{ Tests GetAverageWorkTimeForTask }

TestGetAverageWorkTimeForTask**;**

{ Test for PrintWorkSummaryForPerson }

TestPrintWorkSummaryForPerson**;**

{ Test PrintPersonForTimeTask }

TestPrintPersonForTask**;**

{ Test TestBusiestPerson }

TestBusiestPerson**;**

{ Test GetToalWorkEntryCount }

TestGetToalWorkEntryCount**;**

{ Test DeletePerson }

TestDeletePerson**;**

(\* { Test Performance with work entries }

TestPerformanceWorkEntry;

{ Test Performance with persons and work entries }

TestPerformancePersonWorkEntry; \*)

**END.**

# Tests

Folgend sind die Tests angeführt, die die implementierten Prozeduren und Funktionen der WorkManagementUnit testen.

## Reset

Diese Tests testen die Prozedur Reset.

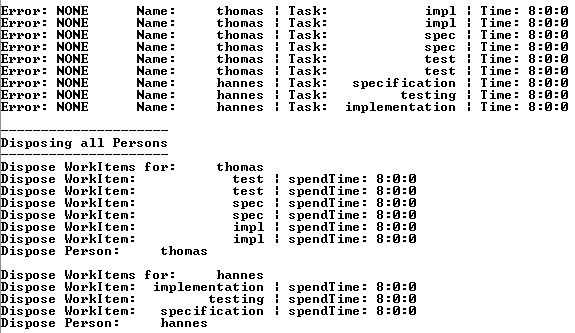
Es wird auch die private Prozedur DisposeWorkItemList getestet, da hier bevor die Personen gelöscht werden, auch ihre erfassten Work Items gelöscht werden müssen. Zu Testzwecken wurden Konsolenausgaben in den Prozeduren Reset und DisposeWorkItemList hinzugefügt damit man sieht das die Einträge in der richtigen Reihenfolge gelöscht werden.

Keine Einträge vorhanden:



Wenn keine Einträge im System vorhanden sind, so können auch keine Dispose Aufrufe erfolgen, daher wird hier zu Testzwecken der oben angeführte String ausgegeben. (Siehe Source)

Einträge vorhanden:



Wenn Einträge im System vorhanden sind, so werden zuerst die Work Items der Work Item List der Person gelöscht bevor die Person aus der Liste gelöscht wird. Dies ist erforderlich, da es sonst zu Speicherleichen führen würde, wenn die Work Items nicht gelöscht werden würden. Das Ankerelement bleibt hierbei bestehen.

## AddWorkEntry

Diese Tests testen die Prozedur AddWorkEntry.

An dieser Schnittstelle wurden keine Änderungen vorgenommen. Der Aufrufer arbeitet weiterhin mit dem Typ WorkEntry. Dieser stellt alle nötigen Informationen bereit damit im System der Eintrag korrekt gespeichert werden kann. Dies begründet sich mit der Zielsetzung, dass es für den Aufrufer keine Änderungen an der bereits existierenden Schnittstelle geben sollte.

Ungültige Zeitangaben:



Wenn ungültige Eingaben erfolgen so wird die dementsprechenden Enumeration auf dem VAR error gesetzt, welche dem Aufrufer über seine Fehleingabe informiert. Diese Einträge werden nicht im System gespeichert. (Siehe Source).

Gültige Eingaben:

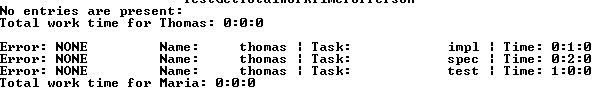


Einträge mit Zeitangaben über einer Minute und unter 8 Stunden werden im System gespeichert und verursachen keine Fehler. Des Weiteren ist anzumerken, dass die Namen der Personen und die Bezeichnungen der Task mit Lower Case gespeichert werden, damit Einträge mit denselben Namen aber unterschiedlicher Groß- und Kleinschreibung nicht als verschiedene Einträge gehandhabt werden.

## GetTotalWorkTimeForPerson

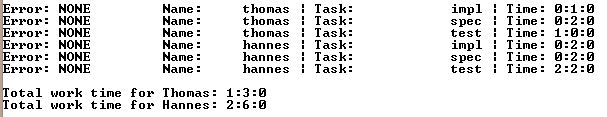
Diese Tests testen die Prozedur GetTotalWorkTimeForPerson.

Keine Einträge vorhanden:



Wenn keine Einträge im System vorhanden sind, so wird weiterhin eine TimeSpan Instanz mit allen Zeitwerten auf 0 gesetzt ausgegeben, das es weiterhin keine Einträge unter einer Minute im System geben kann.

Einträge vorhanden:



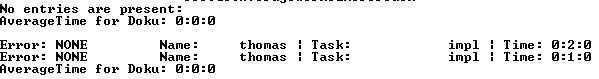
Wenn Einträge im System vorhanden sind, so werden weiterhin die korrekten Ergebnisse ausgeben.

## GetAverageWorkTimeForTask

Diese Tests testen die Prozedur GetAverageWorkTimeForTask.

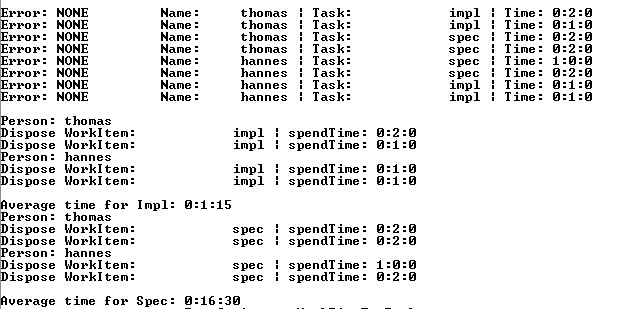
Da hier die Work Items der einzelnen Personen kopiert werden, wurden die Testausgaben für das DisposeWorkItemList wieder eingeführt, um zu zeigen, dass die kopierten Einträge ordnungsgemäß gelöscht werden.

Keine Einträge vorhanden:



Wenn keine Einträge vorhanden sind, so wird weiterhin eine TimeSpan Instanz zurückgegeben, deren Zeitangaben alle auf 0 gesetzt sind, da es auch hier keinen Eintrag geben kann, der kleiner als eine Minute ist, wenn Einträge vorhanden wären.

Einträge vorhanden oder nicht gefunden:

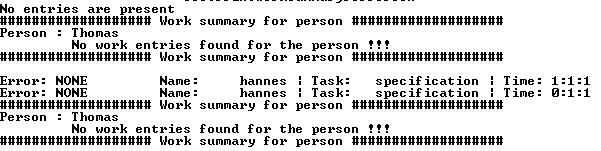


Sind Einträge im System vorhanden so werden die korrekten Durchschnittswerte berechnet (ACHTUNG Ganzzahlige Division). Da in dieser Prozedur die Work Items der verschiedenen Personen kopiert werden, müssen diese auch nachdem sie nicht mehr verwendet werden, gelöscht werden, was über die Konsolenausgaben ersichtlich ist.

## PrintWorkSummaryForPerson

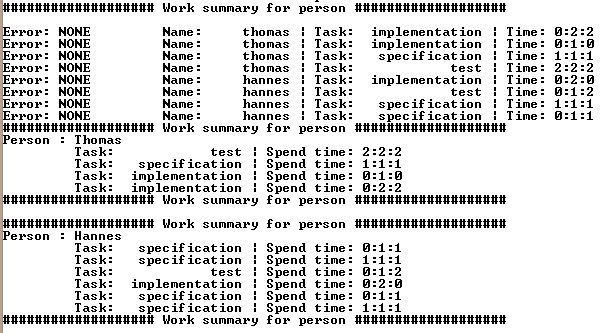
Diese Test testen die Prozedur PrintWorkSummaryForPerson.

Keine Einträge vorhanden:



Sollten keine Einträge im System vorhanden sein, so wird dies über die Konsole angezeigt.

Einträge vorhanden:

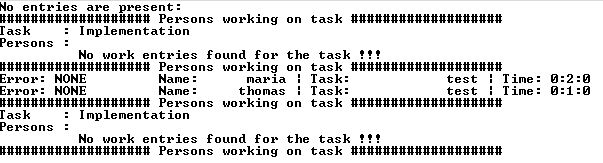


Es ist zu sehen, dass die Einträge in verkehrter Reihenfolge ausgegeben werden, als sie eingespielt wurden. Dies liegt daran, dass die Einträge immer am Beginn der Liste angefügt werden, um die Performance zu verbessern. Da es keine besondere Anforderung gab, in welcher Reihenfolge die Einträge ausgeben werden müssen, wurde auch keine InvertList Prozedur implementiert.

## PrintPersonForTask

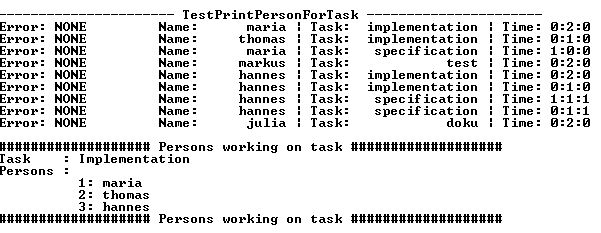
Diese Test testen die Prozedur PrintPersonForTask.

Keine Einträge vorhanden:



Sollten keine Einträge vorhanden sein, so wird dies über die Konsole angezeigt.

Einträge vorhanden:



Sind Einträge vorhanden, so werden die Personen, die an einen Task arbeiten ausgegeben. Auch hier wird die Bezeichnung des Task mit ignore case behandelt.

## BusiestPerson

Diese Tests testen die Funktion BusiestPerson.

Keine Einträge vorhanden:



Wenn keine Einträge im System vorhanden sind so wird ein Leerstring zurückgegeben.

Einträge vorhanden:

Wenn es Einträge im System gibt, dann gibt es zwei Möglichkeiten:

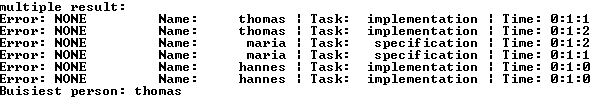
1. Es gibt ein eindeutiges Resultat
2. Es gibt mehrere Resultate

Eindeutiges Resultat:



Wenn es nur ein Resultat gibt, dann wird diese Person ausgegeben.

Mehrere Resultate:



Sollte es kein eindeutiges Ergebnis geben, so wird, laut Implementierung, das erste Resultat zurückgeliefert. Da es sich hier aber nicht um das erste Resultat handelt ist wie folgt zu erklären.

Da in der Liste WorkItemList die Einträge vorne eingefügt werden (Performance), ist die Liste natürlich gespiegelt, daher ist eigentlich das letzte Resultat, welches zurückgegeben wird.

## GetTotalWorkEntryCount

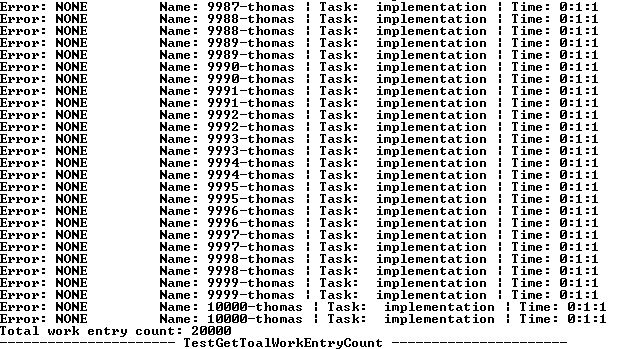
Dieser Test testet die Funktion GetTotalWorkEntryCount.

Keine Einträge vorhanden:



Wenn keine Einträge im System vorhanden sind, so wird als Ergebnis 0 zurückgeliefert.

Einträge vorhanden:

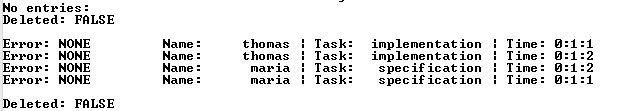


Sollten Einträge im System vorhanden sein, so werden alle Einträge der WorkItemList aller Personen gezählt und das Ergebnis zurückgeliefert.

## DeletePerson

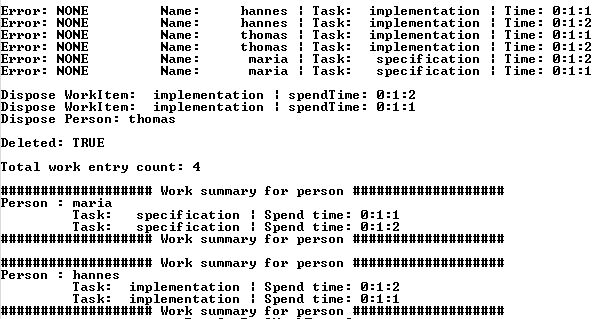
Diese Tests testen die Funktion DeletePerson.

Keine Einträge vorhanden:



Wenn keine Einträge im System vorhanden sind, so wird auch keine Person aus dem System gelöscht, was über den Rückgabewert FALSE angezeigt wird.

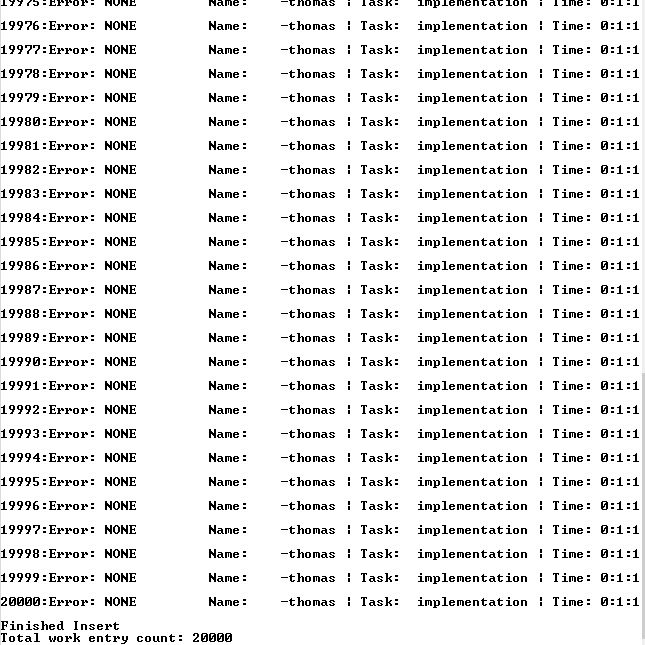
Einträge vorhanden:



Wenn Einträge für eine Person gefunden wurden, so werden zuerst die Work Item der WorkEntryList der Person gelöscht und anschließend die Person selbst. Zu Testzwecken wurden Konsolenausgaben eingeführt, die zeigen das die Work Item und die Person tatsächlich gelöscht wurden (Siehe Source). Ebenso wurde zu Testzwecken die verbliebene Anzahl der Einträge sowie eine Summary der verbliebenen Personen eingeführt, um zu zeigen das die Liste noch funktioniert und nur die gewählte Person samt ihrer Einträge gelöscht wurde.

## Performance (WorkItem)

Dieser Test testet das Verhalten wenn viele Einträge für ein und dieselbe Person getätigt werden. Das bedeutet, dass für diese Person, nach ihrem Anlegen, jeweils noch weitere Work Item gespeichert werden.

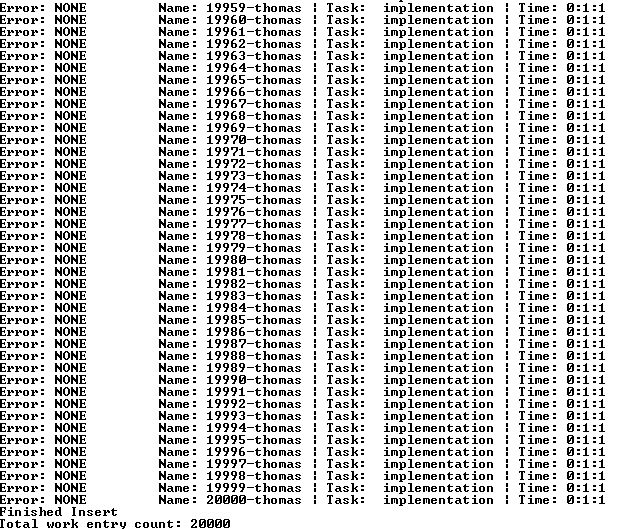


Oben ist ein Auszug aus der Konsole angeführt, für weitere Details sei auf den Source verweisen.

Es ist bei der Konsolenausgabe zu sehen, dass es keine Performance Einbußen gibt. Dies liegt daran, dass die Elemente am Anfang der Liste eingefügt werden.

## Performance (Person)

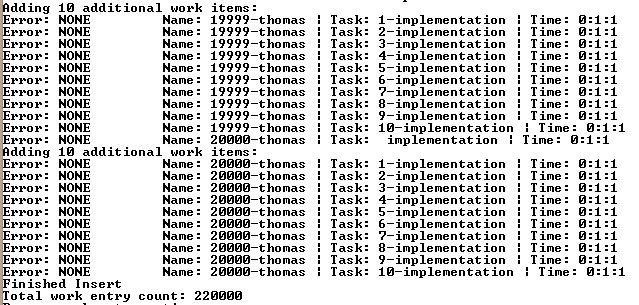
Dieser Test testet die Performance wenn mehrere verschiedene Personen mit jeweils einen Work Item gespeichert werden.



Oben ist ein Auszug aus der Konsole angeführt, für weitere Details sei auf den Source verwiesen. Da es sich bei dem Gedächtnis um eine doppelt verkettete zyklische Liste handelt ist es egal das die Elemente am Ende der Liste eingefügt werden, da hier kein Iterieren über die gesamte Liste notwendig ist um an das letzte Element zu gelangen. Auch hier sind keine Performance Einbußen zu erkennen, wenn mehrere Person hintereinander eingefügt werden.

## Performance (Person + WorkItem)

Dieser Test testet die Performance wenn für jede Person 10 weitere Work Item gespeichert werden.



Oben ist ein Auszug aus der Konsole angeführt, für weitere Details sei auf den Source verweisen.

Auch hier gibt es keine Performance Einbrüche wenn so viele Einträge hintereinander gespeichert werden.

# Diskussion

Obwohl mit diesem Datenmodel insbesondere mit dem gewählten Listentyp, die Implementierung verfeinern hat lassen, wäre meiner Meinung nach eine weiter Splittung anwendbar.

Es sollte der Datentyp für die Personen so abgeändert werden, sodass diese eine Liste von Tasks halten und die Tasks wiederum eine Liste von Work Item. So würde die Hierarchie optimal aufgebaut sein. Mittels Hilfsfunktionen könnten dann einfach die Tasks aus der Liste selektiert werden. Es wäre aber anzumerken, dass alle verwendeten Listen doppelt verkettet Listen mit Ankerelement sein sollten, damit das Problem mit Einfügen an der ersten Stelle in der Liste (Invertierung aber bessere Performance) vermieden werden könnte. So könnten alle Elemente immer an der letzte Stelle eingefügt werden, ohne Performanceeinbußen befürchten zu müssen.