[1 Lösungsansatz 3](#_Toc382505661)

[1.1 Allgemein 3](#_Toc382505662)

[1.2 Single-Linked-List Unit 5](#_Toc382505663)

[1.3 Hashtable (Kollisionsbehandlung verkettet) 5](#_Toc382505664)

[1.4 Hashtable (Kollisionsbehandlung ) 5](#_Toc382505665)

[2 Source 6](#_Toc382505666)

[2.1 PrintUtils 6](#_Toc382505667)

[2.2 PrintUtilsTest 7](#_Toc382505668)

[2.3 HashUtils 8](#_Toc382505669)

[2.4 HashUtilsTest 9](#_Toc382505670)

[2.5 Integer Utils 10](#_Toc382505671)

[2.6 IntegerUtilsTest 11](#_Toc382505672)

[2.7 WorkManagementInterface 12](#_Toc382505673)

[2.8 WorkManagementListUnit 14](#_Toc382505674)

[2.9 WorkManagementListUnitTest 19](#_Toc382505675)

[2.10 WorkManangementHashChainedUnit 23](#_Toc382505676)

[2.11 WorkManangementHashChainedUnitTest 28](#_Toc382505677)

[2.12 WorkManagementHashOpenUnit 32](#_Toc382505678)

[2.13 WorkManagementHashOpenUnitTest 38](#_Toc382505679)

[3 Tests 42](#_Toc382505680)

[3.1 Single-Linked-List Unit 42](#_Toc382505681)

[3.1.1 SetWorkEntryContext 42](#_Toc382505682)

[3.1.2 CreateWorkEntry 43](#_Toc382505683)

[3.1.3 AddWorkEntry 43](#_Toc382505684)

[3.1.4 GetTotalWorkTimeForPerson 45](#_Toc382505685)

[3.1.5 RemovePerson 45](#_Toc382505686)

[3.2 HashChained Unit 46](#_Toc382505687)

[3.2.1 SetWorkEntryContext 46](#_Toc382505688)

[3.2.2 CreateWorkEntry 47](#_Toc382505689)

[3.2.3 AddWorkEntry 47](#_Toc382505690)

[3.2.4 GetTotalWorkTimeForPerson 48](#_Toc382505691)

[3.2.5 RemovePerson 49](#_Toc382505692)

[3.3 HashOpen Unit 50](#_Toc382505693)

[3.3.1 SetWorkEntryContext 50](#_Toc382505694)

[3.3.2 CreateWorkEntry 51](#_Toc382505695)

[3.3.3 AddWorkEntry 51](#_Toc382505696)

[3.3.4 GetToalWorkTimeForPerson 53](#_Toc382505697)

[3.3.5 RemovePerson 54](#_Toc382505698)

[4 Diskussion 55](#_Toc382505699)

# Lösungsansatz

Folgend ist der Lösungsansatz der Übungen 1 angeführt. Für weitere Dokumentation sei auf den Source und die Tests verweisen.

## Allgemein

Da diese Aufgabe bereits im 1 Semester gemacht wurde, soll diese Implementierung als Basis herangezogen und angepasst werden. Hierbei soll die Schnittstelle in eine eigene Datei herausgezogen werden, welche anschließend über den Tag {$I FILENAME} in die Unit inkludiert werden soll. Dabei kopiert der Pascal Compiler den enthaltenen Source in die Source Datei der Unit hinein. Die Kompilierung erfolgt über die Unit, da dieses Interface dann ein Teil der Unit ist. Die Schnittstelle darf nur Code Fragmente enthalten, sodass bei der Inkludierung keine Kompilierfehler auftreten.

**Aufbau Schnittstelle:**

**INTERFACE**

**USES** …**;**

**TYPE**

…

**PROCEDURE** Reset**;**

**Aufbau Unit:**

**Unit** WorkManagementListunit**;**

// Includes the interface code fragments into this source

{$I WorkManagementInterface.pas}

**Implementation**

**Type**

…

**Var**

…

**Procedure** Reset**;**

Mit diesen Ansatz ist es möglich die Schnittstelle nur einmal zu spezifizieren und in den Implementierungen (Units) zu verwenden. Ebenso soll damit vermieden werden, das Interface zu duplizieren. Es mag vielleicht einen besseren Ansatz geben, aber zurzeit ist mir kein anderer bekannt.

Aber meiner Meinung nach sollte das Interface nicht mehrfach definiert werden, wenn es dasselbe ist und es mehrere Implementierungen dafür gibt.

**Schnittstelle:**

Das Interface soll so implementiert werden, sodass es keine Referenzen auf das verwendete Storage gibt, somit soll es möglich sein verschiedene Implementierungen zur Verfügung zu stellen, die verschiedene Arten von Storages verwenden (Array, Single-Linked-List, Double-Linked-List-With-Anchor, HsahTable,…). Da dies bereits so implementiert wurde, soll hier lediglich die Schnittstelle herausgezogen und so modifiziert werden, sodass alle überflüssigen Funktionen und Prozeduren wegfallen und es um die neuen Funktionalitäten erweitert wird.

**PrintUtils:**

Weil bei den Tests und auch innerhalb der Implementierungen Konsolenausgaben erfolgen, soll ebenfalls eine PrintUtils Unit implementiert werden, mit der formatierte Texte ausgegeben werden können wie z.B.: Header, Error usw. Hierbei wurde auf die Dokumentation von Tests verzichtet, da es sich hierbei um eine triviale Implementierung handelt, sehr wohl wurde aber der Test Source dem Projekt hinzugefügt.

**IntegerUtils:**

Da die Berechnung des Hash der Indexrange verwendet wird, und andere Units auch Integer Operationen benötigen, soll eine IntegerUtils Unit implementiert werden, wo alle Utility Funktion und Prozeduren bezüglich Integer zusammengefasst werden sollen.

**HashUtils:**

Da zwei Aufgaben erfordern die Daten in einer Hashtabelle zu speichern, also in zwei unterschiedlichen Units, soll eine HashUtils Unit implementiert werden, die Algorithmen zum Ermitteln von Hashwerten für Strings bereitstellen. Auch diese kann in anderen Units verwendet werden. Vorerst soll eine Funktion implementiert werden, welche nur die Ordinalwerte der Zeichen für die Berechnung des Hash verwendet um bewusst Kollisionen zu produzieren. Da diese Funktion innerhalb einer eigenen Unit gehalten wird, kann diese auch in späterer Folge verbessert werden, bzw. zusätzliche Hashalgorithmen implementiert werden.

**Kontext:**

In den Implementierungen soll es möglich sein, dass der Benutzer einen Kontext definieren kann, welcher die Storage Größe, die Minimumzeitspanne sowie Maximumzeitspanne definieren kann. Diese definierten Werte sollen in der Implementierung zur Validierung verwendet werden. Die Größe soll die Anzahl der zu speichernden Personen definieren. Zu Testzwecken soll die Größe der Hashtabelle der Maximumanzahl der möglich zu speichernden Personen gewählt werden, obwohl es vernünftiger wäre die Anzahl größer zu wählen um einen größeren Adressraum für die Berechneten Indizes zu erhalten.

## Single-Linked-List Unit

Bei dieser Implementierung soll die bereits bestehende Implementierung herangezogen werden, die dieses Storage verwendet. Hierbei besteht der Arbeitsaufwand in Copy Paste der bestehenden Prozeduren und Funktionen. Lediglich das Löschen aller Einträge von einer bestimmten Person muss noch implementiert werden. Als Interface soll das oben beschriebene verwendet und ausgelagert werden. Hierbei soll die Anzahl der zu speichernden Person durch den Kontext beschränkt werden können.

## Hashtable (Kollisionsbehandlung verkettet)

Bei dieser Implementierung soll eine Hashtabelle als Storage verwendet werden, wobei die Kollisionsbehandlung über Verkettung der Elemente mit demselben Hashwert erfolgen soll. Es soll eine Hashtabelle verwendet werden, wobei die Möglichkeit bestehen muss dessen Länge dynamisch durch setLength(array, 10) (vorheriges Löschen der Elemente erforderlich) ändern zu können.. Hierbei soll die vorherige Implementierung herangezogen und angepasst werden, da sich hier nur die Art des Storage ändert und nicht die Funktionalität selbst.

## Hashtable (Kollisionsbehandlung )

Bei dieser Implementierung soll eine Hashtabelle als Storage verwendet werden, wobei die Kollisionsbehandlung über offene Adressierung von Elementen mit demselben Hash erfolgen soll. Hierbei soll es möglich sein die Elemente mit demselben Hash nicht nur bis zum Ende der Hashtabelle einzutragen sondern wenn das Ende erreicht wurde, soll versucht werden einen Index am Anfang der Hashtabelle bis zum berechneten Index zu suchen. Sollte kein Index mehr zur Verfügung stehen, so soll dies den Benutzer über einen Fehler bekanntgegeben werden. Hierbei soll die vorherige Implementierung herangezogen und angepasst werden, da sich auch hier nur die Art des Storage ändert und nicht die Funktionalität selbst. Auch hier soll die Anzahl der zu speichernden Personen beschränkbar sein.

# Source

Folgend ist der Source der implementierten Units sowie aller verwendeten und implementierten Utility Units, sowie deren Tests angeführt.

## PrintUtils

Folgend ist der Source der PrintUtils Unit angeführt, welche Funktionen und Prozeduren zur Verfügung stellt, die eine formatierte Ausgabe von Texten in der Konsole erlauben.

{

This unit provides utility functions and procedure which allows

printing formatted text to the console.

}

**Unit** PrintUtils**;**

**interface**

**Uses** Crt**;**

{

Prints a string as a header with default background and text colour set.

@see

rintHeader(message: String; backgroundColor, testColor: Byte)

@see

Const section for default colour

}

**Procedure** PrintHeader**(message:** **String);**

{

Prints a string as a error with default background and text colour set.

@see

rintHeader(message: String; backgroundColor, testColor: Byte)

@see

Const section for default colour

}

**Procedure** PrintError**(message:** **String);**

{

Prints a string with the defined formatting settings.

@param

background: the background colour

@param

text: the text colour to be set

}

**Procedure** Print**(message:** **String;** background**,** text**:** Byte**);**

**implementation**

**Const**

DEFAULT\_HEADER\_BACKGROUN\_COLOR**:** Byte **=** Blue**;**

DEFAULT\_HEADER\_TEXT\_COLOR**:** Byte **=** White**;**

DEFAULT\_ERROR\_BACKGROUN\_COLOR**:** Byte **=** Red**;**

DEFAULT\_ERROR\_TEXT\_COLOR**:** Byte **=** White**;**

**Procedure** PrintHeader**(message:** **String);**

**Begin**

Print**(message,** DEFAULT\_HEADER\_BACKGROUN\_COLOR**,** DEFAULT\_HEADER\_TEXT\_COLOR**);**

**End;**

**Procedure** PrintError**(message:** **String);**

**Begin**

Print**(message,** DEFAULT\_ERROR\_BACKGROUN\_COLOR**,** DEFAULT\_ERROR\_TEXT\_COLOR**);**

**End;**

**Procedure** Print**(message:** **String;** background**,** text**:** Byte**);**

**Begin**

TextColor**(**text**);**

TextBackground**(**background**);**

write**(message);**

NormVideo**();**

writeln**;**

**End;**

**Begin**

**End.**

## PrintUtilsTest

Folgend ist der Source der PrintUtilsTest angeführt, welcher die PrintUtils Unit testet. Es wurde auf eine genauere Beschreibung der Tests verzichtet, es sei hier auf den Source verwiesen.

{

This program tests the PrintUtils unit.

}

**Program** PrintUtilsTest**;**

**Uses**

PrintUtils**,** Sysutils**,** Crt**;**

{

Tests the function PrintHeader

}

**Procedure** TestPrintHeader**;**

**Begin**

// Print header in default colour

PrintUtils**.**PrintHeader**(**'This header should be blue'**);**

**End;**

{

Tests the function PrintError

}

**Procedure** TestPrintError**;**

**Begin**

// Print header in default colour

PrintUtils**.**PrintError**(**'This error should be red'**);**

**End;**

{

Tests the function Print

}

**Procedure** TestPrint**;**

**Begin**

// Print header in default colour

PrintUtils**.**Print**(**'This test should be white with green background'**,** Green**,** White**);**

writeln**;**

// Print header in default colour

PrintUtils**.**Print**(**'This test should be yellow with blue background'**,** Blue**,** Yellow**);**

writeln**;**

**End;**

**Begin**

// Test PrintHeader

TestPrintHeader**;**

writeln**;**

// Test PrintError

TestPrintError**;**

writeln**;**

// Test Print

TestPrint**;**

writeln**;**

**End.**

## HashUtils

Folgend ist der Source der HashUtils Unit angeführt, welche Utility Funktionen und Prozeduren bereitstellt, die Hashwerte berechnen.

**Unit** HashUtils**;**

**Interface**

**Uses**

IntegerUtils**;**

**Type**

HashState **=** **(**NONE**,** OK**,** INVALID\_RANGE**,** EMPTY\_STRING**);**

StringRange **=** 1**..**255**;**

{

Computes a has value which is the key for the given string value.

The hash key will be calculated via the ordinal value of the given string value.

Be aware that this algorithm causes multiple hash values for different string values

which is caused by the usage of the the ordinal value of the char without the index

in the character sequence.

In case of an error the returned hash will be an undefined index and is not supposed to be used.

@param

value: the string value to be hashed

@param

minIndex: the minimum index of the index range

@param

maxIndex: the maximum index of the index range

@return

the computed hash

the current word state,

INVALID\_RANGE if minIndex overflows maxIndex

OK if hash could be computed without error

}

**Function** ComputeHashOrdinalOnly**(**value**:** **String;** minIndex**,** maxIndex**:** LongInt**;** **VAR** state**:** HashState**):** Longint**;**

**Implementation**

**Function** ComputeHashOrdinalOnly**(**value**:** **String;** minIndex**,** maxIndex**:** LongInt**;** **VAR** state**:** HashState**):** Longint**;**

**Var**

hash**,** range**:** LongInt**;**

i**:** StringRange**;**

**Begin**

state **:=** HashState**.**OK**;**

hash **:=** minIndex **-** 1**;**

// Invalid range of index

**if** **(not** IntegerUtils**.**IsValidRange**(**minIndex**,** maxIndex**,** true**))** **then** **begin**

state **:=** HashState**.**INVALID\_RANGE**;**

writeln**(**'range error'**);**

**end**

// EMpty String

**else** **if** **(**Length**(**value**)** **=** 0**)** **then** **begin**

state **:=** HashState**.**EMPTY\_STRING**;**

**end**

**else** **begin**

hash **:=** 0**;**

range **:=** maxIndex **-** minIndex **+** 1**;**

**for** i **:=** Low**(**StringRange**)** **to** Length**(**value**)** **do** **begin**

hash **:=** **(**hash **+** ORD**(**value**[**i**]))** **MOD** range**;**

**end;**

{ Avoids zweo has index }

ComputeHashOrdinalOnly **:=** hash **+** minIndex**;**

**end;**

**End;**

**Begin**

**End.**

## HashUtilsTest

Folgend ist der Source der HashUtilsTest angeführt, welcher die HashUtils Unit testet. Auf eine genauere Beschreibung der Tests wurde verzichtet und es sei hierbei auf den Source verweisen.

{

This program tests the HashUtils unit.

}

**Program** HashUtilsTest**;**

**Uses**

HashUtils**,** PrintUtils**,** Crt**,** Sysutils**;**

**Type**

IndexRange **=** 1**..**100**;**

{

Tests the function ComputeHashOrdinalOnly.

}

**Procedure** TestComputeHashOrdinalOnly**;**

**Var**

hash**:** IndexRange**;**

state**:** HashState**;**

i**:** Integer**;**

**Begin**

// Produces same hash value

PrintUtils**.**PrintHeader**(**'Produces equal hash value'**);**

**for** i **:=** 1 **to** 2 **do** **begin**

writeln**(**i**,** '. Thomas: '**,** HashUtils**.**ComputeHashOrdinalOnly**(**'Thomas'**,** Low**(**IndexRange**),** High**(**IndexRange**),** state**));**

writeln**(**i**,** '. State : '**,** state**);**

writeln**(**i**,** '. Hugo : '**,** HashUtils**.**ComputeHashOrdinalOnly**(**'Hugo'**,** Low**(**IndexRange**),** High**(**IndexRange**),** state**));**

writeln**(**i**,** '. State : '**,** state**);**

writeln**;**

**end;**

writeln**;**

// Produces collisions

PrintUtils**.**PrintHeader**(**'Produces collsions'**);**

writeln**(**'Thomas: '**,** HashUtils**.**ComputeHashOrdinalOnly**(**'Thomas'**,** Low**(**IndexRange**),** High**(**IndexRange**),** state**));**

writeln**(**'State : '**,** state**);**

writeln**;**

writeln**(**'Wolfganf: '**,** HashUtils**.**ComputeHashOrdinalOnly**(**'Wolfganf'**,** Low**(**IndexRange**),** High**(**IndexRange**),** state**));**

writeln**(**'State : '**,** state**);**

**End;**

**Begin**

// Test ComputeHashOrdinalOnly

PrintUtils**.**Print**(**'Test ComputeHashOrdinalOnly'**,** Green**,** White**);**

TestComputeHashOrdinalOnly

**End.**

## Integer Utils

Folgend ist der Source der Unit IntegerUtils angeführt, welche implementiert wurde um Hilfsfunktionen und Prozeduren zur Verfügung zu stellen, welche das Handling mit Integer erleichtern und in anderen Programmen und Units wiederverwendet werden kann.

{

This unit provides utility function and procedures for handling integer type.

}

**Unit** IntegerUtils**;**

**Interface**

{

Validate the two given values if the represent a valid range.

@param

min: the minimum value of the range

@param

max: the maximum value of the range

@return

true if the two values represent a valid range or if they are equal,

false otherwise

@see

IsValidRange(min, max: Integer; strict: Boolean): Boolean; overload;

}

**Function** IsValidRange**(**min**,** max**:** Integer**):** Boolean**;** **overload;**

{

Validate the two given values if the represent a valid range.

@param

min: the minimum value of the range

@param

max: the maximum value of the range

@param

strict: true if the minimum must be smaller than the maximum

@return

true if the two values represent a valid range,

false otherwise

}

**Function** IsValidRange**(**min**,** max**:** Integer**;** strict**:** Boolean**):** Boolean**;** **overload;**

**Implementation**

**Function** IsValidRange**(**min**,** max**:** Integer**):** Boolean**;**

**Begin**

IsValidRange **:=** IsValidRange**(**min**,** max**,** false**);**

**End;**

**Function** IsValidRange**(**min**,** max**:** Integer**;** strict**:** Boolean**):** Boolean**;**

**Begin**

**if** **(**strict**)** **then** **begin**

IsValidRange **:=** min **<** max**;**

**end**

**else** **begin**

IsValidRange **:=** min **<=** max**;**

**end;**

**End;**

**Begin**

**End.**

## IntegerUtilsTest

Folgend ist der Source der IntegerUtilsTest angeführt, welche die Unit IntegerUtils testet. Auf eine genauere Beschreibung der Tests wurde verzichtet und es sei hierbei auf den Source verweisen.

{

This program tests the IntegerUtils unit.

}

**Program** IntegerUtilsTest**;**

**Uses**

Sysutils**,** Crt**,** IntegerUtils**,** PrintUtils**;**

{

Tests the function IsValidRange with no strict option

}

**procedure** TestIsValidRangeNotStrict**;**

**Var**

min**,** max**:** Integer**;**

**Begin**

// Equal

PrintUtils**.**PrintHeader**(**'Equal min, max'**);**

min **:=** 55**;**

max **:=** 55**;**

writeln**(**'min: '**,** min**,** ' / max: '**,** max**,** ' / result: '**,** IntegerUtils**.**IsValidRange**(**min**,** max**));**

writeln**;**

// Max overflow

PrintUtils**.**PrintHeader**(**'max overflow'**);**

min **:=** 55**;**

max **:=** 10**;**

writeln**(**'min: '**,** min**,** ' / max: '**,** max**,** ' / result: '**,** IntegerUtils**.**IsValidRange**(**min**,** max**));**

writeln**;**

// Valid

PrintUtils**.**PrintHeader**(**'Valid'**);**

min **:=** 10**;**

max **:=** 55**;**

writeln**(**'min: '**,** min**,** ' / max: '**,** max**,** ' / result: '**,** IntegerUtils**.**IsValidRange**(**min**,** max**));**

writeln**;**

**End;**

{

Tests the function IsValidRange with strict option

}

**procedure** TestIsValidRangeStrict**;**

**Var**

min**,** max**:** Integer**;**

**Begin**

// Equal not strict

PrintUtils**.**PrintHeader**(**'Equal min, max not strict'**);**

min **:=** 55**;**

max **:=** 55**;**

writeln**(**'min: '**,** min**,** ' / max: '**,** max**,** ' / result: '**,** IntegerUtils**.**IsValidRange**(**min**,** max**,** false**));**

writeln**;**

// Equal strict

PrintUtils**.**PrintHeader**(**'Equal min, max strict'**);**

min **:=** 55**;**

max **:=** 55**;**

writeln**(**'min: '**,** min**,** ' / max: '**,** max**,** ' / result: '**,** IntegerUtils**.**IsValidRange**(**min**,** max**,** true**));**

writeln**;**

// Max overflow

PrintUtils**.**PrintHeader**(**'max overflow'**);**

min **:=** 55**;**

max **:=** 10**;**

writeln**(**'min: '**,** min**,** ' / max: '**,** max**,** ' / result: '**,** IntegerUtils**.**IsValidRange**(**min**,** max**,** true**));**

writeln**;**

// Valid

PrintUtils**.**PrintHeader**(**'Valid'**);**

min **:=** 10**;**

max **:=** 55**;**

writeln**(**'min: '**,** min**,** ' / max: '**,** max**,** ' / result: '**,** IntegerUtils**.**IsValidRange**(**min**,** max**,** true**));**

writeln**;**

**End;**

**Begin**

// Test IsValidRange not strict

PrintUtils**.**Print**(**'Tests IsValidRange not strict'**,** Green**,** White**);**

TestIsValidRangeNotStrict**;**

writeln**;**

// Test IsValidRange strict

PrintUtils**.**Print**(**'Tests IsValidRange strict'**,** Green**,** White**);**

TestIsValidRangeStrict**;**

writeln**;**

**End.**

## WorkManagementInterface

Dieser Source stellt das Interface dar, welches in die verschiedenen Units als Datei eingebunden wird. Es ist keine vollständige Source Datei, da sie von Compiler in die Unit hineinkopiert wird.

{

This interface specifies the functions and procedures which a

WorkManagement unit implementation must provide.

This interface does not specify any used storage, so the implementation

itself must provide the storage and is free to choose the type of storage.

}

**Interface**

// Uses the time span unit for the spend time of a WorkEntry

**Uses** sysutils**,** Crt**,** TImeSpanUnit**,** PrintHandler**;**

// The types which are visible and usable for the caller

**Type**

// The state codes which provide information of the current state

StateCode **=** **(**OK**,** INVALID\_SPAN**,** STORAGE\_FULL**,** PERSON\_NOT\_FOUND**,** INVALID\_CONTEXT**);**

// Compound representing a work entry

WorkEntryRec **=** **Record**

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

**End;**

// The context which specifies the behaviour of this module

WorkEntryContext **=** **Record**

minimumSpan**:** TimeSpan**;**

maximumSpan**:** TimeSpan**;**

storageSize**:** Longint**;**

**End;**

{

Cleans the storage.

}

**Procedure** CleanStorage**;**

{

Sets the context for the module.

@param

minimumSpan: the minimumSpan which a WorkEntry instance is allowed have.

@param

maximumSpan the maximum TimeSpan a WorkEntry instance is allowed to have.

@param

storageSize:: the allowed size of the backed storage

}

**Procedure** setWorkEntryContext**(**minimumSpan**,** maximumSpan**:** TimeSpan**;** storageSize**:** Longint**;** **Var** state**:** StateCode**);**

{

Creates a new WorkEntry for the given data.

@param

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

@param

state: The state of the procedure work.

@return

the created WorkEntry instance

}

**Function** CreateWorkEntry**(**spendTime**:** TimeSpan**;** **VAR** state**:** StateCode**):** WorkEntryRec**;**

{

Adds a WorkEntry to the backed storage.

@param

name: the name of the person where this WorkEntry belongs to.

@param

entry: the WorkEntry to be added

@error

the state code which defines the state of the done work:

OK: If no error occurs

INVALID\_SPAN: If the spendTime field is an invalid time span

STORAGE\_FULL: If the storage is already full

}

**Function** AddWorkEntry**(**name**:** **String;** entry**:** WorkEntryRec**):** StateCode**;**

{

Gets the total work time for the given person.

@param

name: the name of the person

@param

span: 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

@return

the state code which defines the state of the done work

OK: If the entries could be deleted

PERSON\_NOT\_FOUND: If the person could not be found

}

**Procedure** GetTotalWorkTimeForPerson**(**name**:** **String;** **Var** span**:** TimeSpan**;** **Var** state**:** StateCode**);**

{

Removes the given person.

@param

name: the name of the person to deleted WorkEntry entries from

@return

the state code which defines the state of the done work

OK: If the WorkEntry entries could be deleted

PERSON\_NOT\_FOUND: If the person could not be found

}

**Function** RemovePerson**(**name**:** **String):** StateCode**;**

## WorkManagementListUnit

Dieser Source stellt die Implementierung der WorkManagementInterface dar, welches als Storage eine Single-Connected-Linked-List verwendet.

{

This implementation implements the interface WorkManagementInterface and uses

a Single-Connected-Linked-list as the storage.

The default storage size is set to: 10.

The default minimum TimeSpan is set to: 0:1:0

The default maximum TimeSpan is set to: 8:0:0

}

**Unit** WorkManagementListUnit**;**

{$I WorkManagementInterface.pas}

// For testing, must be removed when used productive

// ################ For testing ################

**procedure** PrintPersons**;**

// ################ For testing ################

**Implementation**

**Type**

// Pointer to the WorkEntryNodeRec

WorkEntryNode **=** **^**WorkEntry**;**

// Pointer to the WorkEntryNode used as separate tpye for list

WorkEntryList **=** WorkEntryNode**;**

// WorkEntry which hold WorkEntryRec which gets used for list node

WorkEntry **=** **Record**

workEntry**:** WorkEntryRec**;**

next**:** WorkEntryNode**;**

**End;**

// Pointer to the PersonEntryRec type

PersonEntry **=** **^**PersonEntryRec**;**

// The pointer to the PersonEntryRec used for the list

PersonEntryList **=** PersonEntry**;**

// The person entry record which holds a work entry

PersonEntryRec **=** **Record**

name**:** **String;**

entries**:** WorkEntryList**;**

next**:** PersonEntry**;**

**end;**

{

Unit visible variables

}

**Var**

// Single connected list used as storage

storage**:** PersonEntryList**;**

// The context of this module

context**:** WorkEntryContext**;**

// Internal State holder

state**:** StateCode**;**

// current size of the list

size**:** Longint**;**

**Const**

DEFAULT\_STORAGE\_SIZE **=** 1**;**

// ################# Private function and procedure #################

{

Creates a person entry instance. Be aware that this method returns a pointer and you will need to

dispose this instance manually.

@param

name: the name of the person

@param

entries: the WorkEntryList instance which will be set on the person entry

@return

the created PersonEntry instance

}

**Function** CreatePersonEntry**(**name**:** **String;** entries**:** WorkEntryList**):** PersonEntry**;**

**Begin**

// Create PersonEntry

CreatePersonEntry **:=** New**(**PersonEntry**);**

CreatePersonEntry**^.**name **:=** LowerCase**(**name**);**

CreatePersonEntry**^.**entries **:=** entries**;**

CreatePersonEntry**^.**next **:=** **nil;**

**End;**

{

Creates a WorkEntryNode for the given workEntryRec

@param

entry: the WorkEntryRec to create WorkEntryNode

@return

the created WorkEntryNode

}

**Function** CreateWorkEntryNode**(**entry**:** WorkEntryRec**):** WorkEntryNode**;**

**Begin**

CreateWorkEntryNode **:=** New**(**WorkEntryNode**);**

CreateWorkEntryNode**^.**workEntry **:=** entry**;**

**End;**

{

Gets the PersonEntry by its name from the storage.

@param

name: the name of the person to be searched

@return

the found PersonEntry, notherwise

}

**Function** GetPersonEntry**(**name**:** **String):** PersonEntry**;**

**Var**

node**:** PersonEntry**;**

upperName**:** **String;**

**Begin**

node **:=** storage**;**

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

**while** **((**node **<>** **nil)** **and** **(**node**^.**name **<>** upperName**))do** **begin**

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

**end;**

GetPersonEntry **:=** node**;**

**End;**

{

Removes a person by deleting all of its contained WorkEntryNodes

and the person itself. The caller must ensure that the next component

is properly connected to the previous of this person, otherwise

the list will be broken.

}

**Procedure** RemovePersonEntry**(**person**:** PersonEntry**);**

**Var**

pred**,** succ**:** WorkEntryList**;**

**Begin**

**if** **(**person **<>** **nil)** **then** **begin**

pred **:=** person**^.**entries**;**

succ **:=** **nil;**

**while** **(**pred **<>** **nil)** **do** **begin**

succ **:=** pred**^.**next**;**

Dispose**(**pred**);**

{ WriteLn('WorkEntry disposed'); }

pred **:=** succ**;**

**end;**

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

{ WriteLn('PersonEntry disposed'); }

size **:=** size **-** 1**;**

**end;**

**End;**

{

Validates if the given TimeSpan is within context borders.

@param

span: the TimeSpan instance to validate

@return

true if the given TimeSpan instance is within borders, false otherwise

}

**Function** IsValidTimeSpan**(**spendTime**:** TimeSpan**):** Boolean**;**

**Begin**

IsValidTImeSpan **:=** **((**TimeSpanToSeconds**(**spendTime**).**error **=** ''**)** **and** **(**TimeSpanUnit**.**TimeSpanToSeconds**(**spendTime**).**timeInSeconds **>=** TimeSpanUnit**.**TimeSpanToSeconds**(**context**.**minimumSpan**).**timeInSeconds**)** **and** **(**TimeSpanUnit**.**TimeSpanToSeconds**(**spendTime**).**timeInSeconds **<=** TimeSpanUnit**.**TimeSpanToSeconds**(**context**.**maximumSpan**).**timeInSeconds**))**

**End;**

// ################# Interface function and procedure #################

**Procedure** SetWorkEntryContext**(**minimumSpan**,** maximumSpan**:** TimeSpan**;** storageSize**:** Longint**;** **Var** state**:** StateCode**);**

**Begin**

state **:=** StateCode**.**OK**;**

// set error if context definition is invalid

**if** **((**TimeSpanUnit**.**TimeSpanToSeconds**(**minimumSpan**).**error **<>** ''**)** **or** **(**TimeSpanUnit**.**TimeSpanToSeconds**(**maximumSpan**).**error **<>** ''**)** **or** **(**TimeSpanUnit**.**IsShorterThan**(**minimumSpan**,** maximumSpan**)** **=** false**)** **or** **(**storageSize **<** DEFAULT\_STORAGE\_SIZE**))** **then** **begin**

state **:=** StateCode**.**INVALID\_CONTEXT**;**

**end**

// set context if valid definition

**else** **begin**

CleanStorage**;**

context**.**minimumSpan **:=** minimumSpan**;**

context**.**maximumSpan **:=** maximumSpan**;**

context**.**storageSize **:=** storageSize**;**

PrintHandler**.**PrintHeader**(**'Context successfully set'**);**

**end;**

writeln**(**'minimumSpan: '**,** TimeSpanUnit**.**TimeSpanToString**(**minimumSpan**));**

writeln**(**'maximumSpan:'**,** TimeSpanUnit**.**TimeSpanToString**(**maximumSpan**));**

writeln**(**'storageSize: '**,** storageSize**);**

writeln**;**

**End;**

{ Cleans the storage }

**Procedure** CleanStorage**;**

**Var**

succ**:** PersonEntry**;**

**Begin**

succ **:=** **nil;**

**while** **(**storage **<>** **nil)** **do** **begin**

succ **:=** storage**^.**next**;**

RemovePersonEntry**(**storage**);**

storage **:=** succ**;**

**end;**

size **:=** 0**;**

**End;**

{ Creates a WorkEntry instance }

**Function** CreateWorkEntry**(**spendTime**:** TimeSpan**;** **VAR** state**:** StateCode**):** WorkEntryRec**;**

**Begin**

state **:=** StateCode**.**OK**;**

**if** **not** IsValidTimeSpan**(**spendTime**)** **then** **begin**

state **:=** StateCode**.**INVALID\_SPAN**;**

writeln**(**'invalid'**);**

CreateWorkEntry**.**spendTime **:=** context**.**minimumSpan**;**

**end**

**else** **begin**

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

**end;**

**End;**

{ Adds a WorkEntry to the backed person or adds a new person if person not found }

**Function** AddWorkEntry**(**name**:** **String;** entry**:** WorkEntryRec**):** StateCode**;**

**Var**

node**:** PersonEntry**;**

workEntries**:** WorkEntryList**;**

entries**:** WorkEntryList**;**

**Begin**

AddWorkEntry **:=** StateCode**.**OK**;**

// Check for invalid TimeSpan set on WorkEntry

**if** **not** IsValidTimeSpan**(**entry**.**spendTime**)** **then** **begin**

AddWorkEntry **:=** StateCode**.**INVALID\_SPAN**;**

**end**

**else** **begin**

// Add entry to person

entries **:=** CreateWorkEntryNode**(**entry**);**

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

**if** **(**node **=** **nil)** **then** **begin**

node **:=** CreatePersonEntry**(**name**,** entries**);**

// Add person to storage

**if** **(**storage **<>** **nil)** **then** **begin**

// Check for storage overflow

**if** **((**size **+** 1**)** **<** context**.**storageSize**)** **then** **begin**

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

storage **:=** node**;**

size **:=** size **+** 1**;**

writeln**(**'Added: '**,** node**^.**name**);**

**end**

// Overflows storage size

**else** **begin**

AddWorkEntry **:=** StateCode**.**STORAGE\_FULL**;**

**end;**

**end**

**else** **begin**

storage **:=** node**;**

writeln**(**'Added: '**,** node**^.**name**);**

**end;**

**end**

// Only add new WorkEntry

**else** **begin**

workEntries **:=** node**^.**entries**;**

node**^.**entries **:=** entries**;**

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

**end;**

**end;**

**end;**

{ Gets the total work time for the given person }

**Procedure** GetTotalWorkTimeForPerson**(**name**:** **String;** **Var** span**:** TimeSpan**;** **Var** state**:** StateCode**);**

**Var**

person**:** PersonEntry**;**

entry**:** WorkEntryList**;**

sec**:** LONGINT**;**

**Begin**

state **:=** StateCode**.**OK**;**

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

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

// Only get total time when person were found

**if** **(**person **<>** **nil)** **then** **begin**

entry **:=** person**^.**entries**;**

**while** **(**entry **<>** **nil)** **do** **begin**

sec **:=** sec **+** TimeSpanUnit**.**TimeSpanToSeconds**(**entry**^.**workEntry**.**spendTime**).**timeInSeconds**;**

entry **:=** entry**^.**next**;**

**end;**

**end**

**else** **begin**

state **:=** StateCode**.**PERSON\_NOT\_FOUND**;**

**end;**

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

**End;**

{ Removes person from the storage }

**Function** RemovePerson**(**name**:** **String):** StateCode**;**

**Var**

pred**,** succ**:** PersonEntry**;**

lowerName**:** **String;**

**Begin**

RemovePerson **:=** StateCode**.**OK**;**

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

succ **:=** storage**;**

pred **:=** **nil;**

**while** **(**succ **<>** **nil)** **and** **(**succ**^.**name **<>** lowerName**)** **do** **begin**

pred **:=** succ**;**

succ **:=** pred**^.**next**;**

**end;**

**if** **(**succ **<>** **nil)** **then** **begin**

**if** **(**pred **<>** **nil)** **then** **begin**

pred**^.**next **:=** succ**^.**next**;**

**end**

**else** **begin**

storage **:=** succ**^.**next**;**

**end;**

RemovePersonEntry**(**succ**);**

**end;**

**End;**

// For testing, must be removed when used productive

// ################ For testing ################

**procedure** PrintPersons**;**

**Var**

pred**,** succ**:** PersonEntry**;**

entry**:** WorkEntryList**;**

**Begin**

pred **:=** storage**;**

succ **:=** **nil;**

PrintHandler**.**PrintHeader**(**'Storage content'**);**

**while** **(**pred **<>** **nil)** **do** **begin**

succ **:=** pred**^.**next**;**

entry **:=** pred**^.**entries**;**

write**(**pred**^.**name**,** ': '**);**

**while** **(**entry **<>** **nil)** **do** **begin**

write**(**TimeSpanUnit**.**TimeSpanToString**(**entry**^.**workEntry**.**spendTime**),** ', '**);**

entry **:=** entry**^.**next**;**

**end;**

writeln**;**

pred **:=** succ**;**

**end;**

size **:=** 0**;**

**End;**

// ################ For testing ################

{ Initialize module }

**Begin**

PrintHandler**.**PrintHeader**(**'Initializing WorkManagementListUnit'**);**

storage **:=** **NIL;**

size **:=** 0**;**

writeln**(**'storage = nil'**);**

writeln**(**'size = 0'**);**

writeln**(**'Creating default context...'**);**

writeln**;**

SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** DEFAULT\_STORAGE\_SIZE**,** state**);**

// Stop when default context cannot be initialized

**if** **(**state **<>** StateCode**.**OK**)** **then** **begin**

PrintHandler**.**PrintError**(**'Invalid context detected, program will exit'**);**

Halt**;**

**end;**

**End.**

## WorkManagementListUnitTest

Folgend ist der Source der WOrkManagementListUnitTest angeführt welche die WorkManagementListUnit testet.

**Program** WorkManagementListUnitTest**;**

**Uses** sysutils**,** Crt**,** WorkManagementListUnit**,** PrintUtils**,** TimeSpanUnit**;**

**Var**

state**:** StateCode**;**

{

Test the procedure which allows setting of the context.

}

**Procedure** TestSetWorkEntryContext**;**

**Begin**

// Invalid MinimumSpan

PrintUtils**.**PrintHeader**(**'Invalid MinimumSpan'**);**

WorkManagementListUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 60**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**),** 110**,** state**);**

writeln**;**

// Invalid MaximumSpan

PrintUtils**.**PrintHeader**(**'Invalid MaximumSpan'**);**

WorkManagementListUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 1**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 60**),** 110**,** state**);**

writeln**;**

// Invalid Range

PrintUtils**.**PrintHeader**(**'Invalid Range'**);**

WorkManagementListUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 30**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 10**),** 110**,** state**);**

writeln**;**

// Invalid StorageSize

PrintUtils**.**PrintHeader**(**'Invalid Storage size'**);**

WorkManagementListUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 30**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 50**),** 99**,** state**);**

writeln**;**

// Valid context

PrintUtils**.**PrintHeader**(**'Valid context definition'**);**

WorkManagementListUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 30**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 50**),** 100**,** state**);**

**End;**

{

Tests the function CreateWorkEntry

}

**Procedure** TestCreateWorkEntry**;**

**Var**

entry**:** WorkEntryRec**;**

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

state**:** StateCode**;**

**Begin**

WorkManagementListUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 100**,** state**);**

// Invalid TimeSpan

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 60**);**

entry **:=** WorkManagementListUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'Invalid TimeSpan'**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'WorkEntry.spendTime: '**,** TimeSpanUnit**.**TimeSpanToString**(**entry**.**spendTime**));**

writeln**(**'State : '**,** state**);**

writeln**;**

// TimeSpan overflows minimum

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 59**);**

entry **:=** WorkManagementListUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'TimeSpan overflow minimum'**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'WorkEntry.spendTime: '**,** TimeSpanUnit**.**TimeSpanToString**(**entry**.**spendTime**));**

writeln**(**'State : '**,** state**);**

// TimeSpan overflows maximum

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 1**);**

entry **:=** WorkManagementListUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'TimeSpan overflow maximum'**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'WorkEntry.spendTime: '**,** TimeSpanUnit**.**TimeSpanToString**(**entry**.**spendTime**));**

writeln**(**'State : '**,** state**);**

// Valid TimeSpan

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**);**

entry **:=** WorkManagementListUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'Valid TimeSpan'**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'WorkEntry.spendTime: '**,** TimeSpanUnit**.**TimeSpanToString**(**entry**.**spendTime**));**

writeln**(**'State : '**,** state**);**

**End;**

{

Tests the function AddWorkEntry

}

**Procedure** TestAddWorkEntry**;**

**Var**

entry**:** WorkEntryRec**;**

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

state**:** StateCode**;**

i**,** j**:** Integer**;**

**Begin**

// Invalid Time span

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 60**);**

entry**.**spendTime **:=** span**;**

PrintUtils**.**PrintHeader**(**'Invalid TimeSpan '**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** WorkManagementListUnit**.**AddWorkEntry**(**'thomas'**,** entry**));**

writeln**;**

// Overlfows minimum span

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 1**);**

entry**.**spendTime **:=** span**;**

PrintUtils**.**PrintHeader**(**'Overflows minimumSpan '**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** WorkManagementListUnit**.**AddWorkEntry**(**'thomas'**,** entry**));**

writeln**;**

// Overlfows maximum span

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 1**);**

entry**.**spendTime **:=** span**;**

PrintUtils**.**PrintHeader**(**'Overflows maximumSpan '**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** WorkManagementListUnit**.**AddWorkEntry**(**'thomas'**,** entry**));**

writeln**;**

// Overlfows maximum storage

WorkManagementListUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 3**,** state**);**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**);**

entry **:=** WorkManagementListUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'Overflows storage size '**);**

**for** i **:=** 1 **to** 4 **do** **begin**

writeln**(**'Try to add : '**,** 'thomas\_'**,** i**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

state **:=** WorkManagementListUnit**.**AddWorkEntry**(**'thomas\_' **+** IntToStr**(**i**),** entry**);**

writeln**(**'State : '**,** state**);**

writeln**;**

**end;**

writeln**;**

PrintPersons**;**

// Normal behavior

WorkManagementListUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 3**,** state**);**

PrintUtils**.**PrintHeader**(**'Overflows storage size '**);**

**for** i **:=** 1 **to** 3 **do** **begin**

writeln**(**'Try to add : '**,** 'thomas\_'**,** i**);**

**For** j **:=** 1 **to** 5 **do** **begin**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** j**);**

entry **:=** WorkManagementListUnit**.**CreateWorkEntry**(**span**,** state**);**

state **:=** WorkManagementListUnit**.**AddWorkEntry**(**'thomas\_' **+** IntToStr**(**i**),** entry**);**

**end;**

writeln**(**'State : '**,** state**);**

writeln**;**

**end;**

writeln**;**

PrintPersons**;**

**End;**

{

Tests the procedure GetTotalWorkTimeForPerson

}

**Procedure** TestGetTotalWorkTimeForPerson**;**

**Var**

entry**:** WorkEntryRec**;**

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

state**:** StateCode**;**

i**,** j**:** Integer**;**

**Begin**

WorkManagementListUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 10**,** state**);**

**for** i **:=** 1 **to** 11 **do** **begin**

**For** j **:=** 1 **to** 5 **do** **begin**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** j**);**

entry **:=** WorkManagementListUnit**.**CreateWorkEntry**(**span**,** state**);**

state **:=** WorkManagementListUnit**.**AddWorkEntry**(**'thomas\_' **+** IntToStr**(**i**),** entry**);**

**end;**

**end;**

// Person not found

PrintUtils**.**PrintHeader**(**'Person not found'**);**

WorkManagementListUnit**.**GetTotalWorkTimeForPerson**(**'thomas\_11'**,** span**,** state**);**

writeln**(**'Person: '**,** 'Not found'**);**

writeln**(**'Span : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** state**);**

writeln**;**

// Normal bhavor

PrintUtils**.**PrintHeader**(**'Person found'**);**

WorkManagementListUnit**.**GetTotalWorkTimeForPerson**(**'thomas\_10'**,** span**,** state**);**

writeln**(**'Person: '**,** 'thomas\_10'**);**

writeln**(**'Span : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** state**);**

**End;**

{

Test the function RemovePerson

}

**procedure** TestRemovePerson**;**

**Var**

entry**:** WorkEntryRec**;**

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

state**:** StateCode**;**

i**,** j**:** Integer**;**

**Begin**

WorkManagementListUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 10**,** state**);**

**for** i **:=** 1 **to** 5 **do** **begin**

**For** j **:=** 1 **to** 5 **do** **begin**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** j**);**

entry **:=** WorkManagementListUnit**.**CreateWorkEntry**(**span**,** state**);**

state **:=** WorkManagementListUnit**.**AddWorkEntry**(**'thomas\_' **+** IntToStr**(**i**),** entry**);**

**end;**

**end;**

// Person not found

PrintUtils**.**PrintHeader**(**'Person not found in storage'**);**

state **:=** WorkManagementListUnit**.**RemovePerson**(**'thomas\_6'**);**

writeln**(**'Person: '**,** 'thomas\_6'**);**

writeln**(**'State : '**,** state**);**

WorkManagementListUnit**.**PrintPersons**;**

writeln**;**

// First Person removed

PrintUtils**.**PrintHeader**(**'First Person removed'**);**

state **:=** WorkManagementListUnit**.**RemovePerson**(**'thomas\_5'**);**

writeln**(**'Person: '**,** 'thomas\_5'**);**

writeln**(**'State : '**,** state**);**

WorkManagementListUnit**.**PrintPersons**;**

writeln**;**

// Last Person removed

PrintUtils**.**PrintHeader**(**'Last Person removed'**);**

state **:=** WorkManagementListUnit**.**RemovePerson**(**'thomas\_1'**);**

writeln**(**'Person: '**,** 'thomas\_1'**);**

writeln**(**'State : '**,** state**);**

WorkManagementListUnit**.**PrintPersons**;**

writeln**;**

// Middle Person removed

PrintUtils**.**PrintHeader**(**'Person in middle of list'**);**

state **:=** WorkManagementListUnit**.**RemovePerson**(**'thomas\_3'**);**

writeln**(**'Person: '**,** 'thomas\_3'**);**

writeln**(**'State : '**,** state**);**

WorkManagementListUnit**.**PrintPersons**;**

writeln**;**

**End;**

**Begin**

// Test SetWorkEntryContext

PrintUtils**.**Print**(**'Tests for procedure SetWorkEntryContext'**,** Green**,** White**);**

TestSetWorkEntryContext**;**

WorkManagementListUnit**.**CleanStorage**;**

writeln**;**

// Test CreateWorkEntry

PrintUtils**.**Print**(**'Tests for procedure CreateWorkEntry'**,** Green**,** White**);**

TestCreateWorkEntry**;**

WorkManagementListUnit**.**CleanStorage**;**

writeln**;**

// Test AddWorkEntry

PrintUtils**.**Print**(**'Tests for procedure AddWorkEntry'**,** Green**,** White**);**

TestAddWorkEntry**;**

WorkManagementListUnit**.**CleanStorage**;**

writeln**;**

// Test GetTotalWorkTimeForPerson

PrintUtils**.**Print**(**'Tests for procedure GetTotalWorkTimeForPerson'**,** Green**,** White**);**

TestGetTotalWorkTimeForPerson**;**

WorkManagementListUnit**.**CleanStorage**;**

writeln**;**

// Test RemovePerson

PrintUtils**.**Print**(**'Tests for procedure RemovePerson'**,** Green**,** White**);**

TestRemovePerson**;**

WorkManagementListUnit**.**CleanStorage**;**

writeln**;**

**End.**

## WorkManangementHashChainedUnit

Dieser Source stellt die Implementierung der WorkManagementInterface dar, welches als Storage eine Hashtabelle verwendet, welche als Kollisionsbehandlung.

{

This implementation implements the interface WorkManagementInterface and uses

a Single-Connected-Linked-list as the storage.

The default storage size is set to: 10.

The default minimum TimeSpan is set to: 0:1:0

The default maximum TimeSpan is set to: 8:0:0

}

**Unit** WorkManagementHashChainedUnit**;**

{$I WorkManagementInterface.pas}

// For testing, must be removed when used productive

// ################ For testing ################

**procedure** PrintPersons**;**

// ################ For testing ################

**Implementation**

**Uses**

HashUtils**;**

**Type**

// Pointer to the WorkEntryNodeRec

WorkEntryNode **=** **^**WorkEntry**;**

// Pointer to the WorkEntryNode used as separate tpye for list

WorkEntryList **=** WorkEntryNode**;**

// WorkEntry which hold WorkEntryRec which gets used for list node

WorkEntry **=** **Record**

workEntry**:** WorkEntryRec**;**

next**:** WorkEntryNode**;**

**End;**

// Pointer to the PersonEntryRec type

PersonEntry **=** **^**PersonEntryRec**;**

// The pointer to the PersonEntryRec used for the list

PersonEntryList **=** PersonEntry**;**

// The person entry record which holds a work entry

PersonEntryRec **=** **Record**

name**:** **String;**

entries**:** WorkEntryList**;**

next**:** PersonEntry**;**

**end;**

{

Unit visible variables

}

**Var**

// Single connected list used as storage

storage**:** **Array** **of** PersonEntryList**;**

// The context of this module

context**:** WorkEntryContext**;**

// Internal State holder

state**:** StateCode**;**

// current size of the list

size**:** Longint**;**

**Const**

DEFAULT\_STORAGE\_SIZE **=** 1**;**

// ################# Private function and procedure #################

{

Creates a person entry instance. Be aware that this method returns a pointer and you will need to

dispose this instance manually.

@param

name: the name of the person

@param

entries: the WorkEntryList instance which will be set on the person entry

@return

the created PersonEntry instance

}

**Function** CreatePersonEntry**(**name**:** **String;** entries**:** WorkEntryList**):** PersonEntry**;**

**Begin**

// Create PersonEntry

CreatePersonEntry **:=** New**(**PersonEntry**);**

CreatePersonEntry**^.**name **:=** LowerCase**(**name**);**

CreatePersonEntry**^.**entries **:=** entries**;**

CreatePersonEntry**^.**next **:=** **nil;**

**End;**

{

Creates a WorkEntryNode for the given workEntryRec

@param

entry: the WorkEntryRec to create WorkEntryNode

@return

the created WorkEntryNode

}

**Function** CreateWorkEntryNode**(**entry**:** WorkEntryRec**):** WorkEntryNode**;**

**Begin**

CreateWorkEntryNode **:=** New**(**WorkEntryNode**);**

CreateWorkEntryNode**^.**workEntry **:=** entry**;**

CreateWorkEntryNode**^.**next **:=** **nil;**

**End;**

{

Gets the PersonEntry by its name from the storage.

@param

name: the name of the person to be searched

@return

the found PersonEntry, níll otherwise

}

**Function** GetPersonEntry**(**name**:** **String):** PersonEntry**;**

**Var**

node**:** PersonEntry**;**

lowerName**:** **String;**

hash**:** LongInt**;**

state**:** HashState**;**

**Begin**

node **:=** **nil;**

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

hash **:=** HashUtils**.**computeHashOrdinalOnly**(**lowerName**,** Low**(**storage**),** High**(**storage**),** state**);**

**if** **(**state **=** HashState**.**OK**)** **then** **begin**

node **:=** storage**[**hash**];**

**while** **((**node **<>** **nil)** **and** **(**node**^.**name **<>** lowerName**))do** **begin**

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

**end;**

**end;**

GetPersonEntry **:=** node**;**

**End;**

{

Removes a person by deleting all of its contained WorkEntryNodes

and the person itself. The caller must ensure that the next component

is properly connected to the previous of this person, otherwise

the list will be broken.

}

**Procedure** RemovePersonEntry**(**person**:** PersonEntry**);**

**Var**

pred**,** succ**:** WorkEntryList**;**

**Begin**

**if** **(**person **<>** **nil)** **then** **begin**

pred **:=** person**^.**entries**;**

succ **:=** **nil;**

**while** **(**pred **<>** **nil)** **do** **begin**

succ **:=** pred**^.**next**;**

Dispose**(**pred**);**

{ WriteLn('WorkEntry disposed'); }

pred **:=** succ**;**

**end;**

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

{ WriteLn('PersonEntry disposed'); }

size **:=** size **-** 1**;**

**end;**

**End;**

{

Validates if the given TimeSpan is within context borders.

@param

span: the TimeSpan instance to validate

@return

true if the given TimeSpan instance is within borders, false otherwise

}

**Function** IsValidTimeSpan**(**spendTime**:** TimeSpan**):** Boolean**;**

**Begin**

IsValidTImeSpan **:=** **((**TimeSpanToSeconds**(**spendTime**).**error **=** ''**)** **and** **(**TimeSpanUnit**.**TimeSpanToSeconds**(**spendTime**).**timeInSeconds **>=** TimeSpanUnit**.**TimeSpanToSeconds**(**context**.**minimumSpan**).**timeInSeconds**)** **and** **(**TimeSpanUnit**.**TimeSpanToSeconds**(**spendTime**).**timeInSeconds **<=** TimeSpanUnit**.**TimeSpanToSeconds**(**context**.**maximumSpan**).**timeInSeconds**))**

**End;**

// ################# Interface function and procedure #################

**Procedure** SetWorkEntryContext**(**minimumSpan**,** maximumSpan**:** TimeSpan**;** storageSize**:** Longint**;** **Var** state**:** StateCode**);**

**Begin**

state **:=** StateCode**.**OK**;**

// set error if context definition is invalid

**if** **((**TimeSpanUnit**.**TimeSpanToSeconds**(**minimumSpan**).**error **<>** ''**)** **or** **(**TimeSpanUnit**.**TimeSpanToSeconds**(**maximumSpan**).**error **<>** ''**)** **or** **(**TimeSpanUnit**.**IsShorterThan**(**minimumSpan**,** maximumSpan**)** **=** false**)** **or** **(**storageSize **<** DEFAULT\_STORAGE\_SIZE**))** **then** **begin**

state **:=** StateCode**.**INVALID\_CONTEXT**;**

PrintHandler**.**PrintError**(**'Context invalid'**);**

**end**

// set context if valid definition

**else** **begin**

context**.**minimumSpan **:=** minimumSpan**;**

context**.**maximumSpan **:=** maximumSpan**;**

context**.**storageSize **:=** storageSize **+** 1**;**

CleanStorage**;**

SetLength**(**storage**,** context**.**storageSize**);**

PrintHandler**.**PrintHeader**(**'Context successfully set'**);**

**end;**

writeln**(**'minimumSpan: '**,** TimeSpanUnit**.**TimeSpanToString**(**minimumSpan**));**

writeln**(**'maximumSpan:'**,** TimeSpanUnit**.**TimeSpanToString**(**maximumSpan**));**

writeln**(**'storageSize: '**,** storageSize**);**

writeln**;**

**End;**

{ Cleans the storage }

**Procedure** CleanStorage**;**

**Var**

pred**,** succ**:** PersonEntry**;**

i**:** LongInt**;**

**Begin**

**for** i **:=** Low**(**storage**)** **to** High**(**storage**)** **do** **begin**

succ **:=** storage**[**i**];**

pred **:=** **nil;**

**while** **(**succ **<>** **nil)** **do** **begin**

pred **:=** succ**^.**next**;**

RemovePersonEntry**(**succ**);**

succ **:=** pred**;**

**end;**

storage**[**i**]** **:=** **nil;**

**end;**

size **:=** 0**;**

**End;**

{ Creates a WorkEntry instance }

{ Creates a WorkEntry instance }

**Function** CreateWorkEntry**(**spendTime**:** TimeSpan**;** **VAR** state**:** StateCode**):** WorkEntryRec**;**

**Begin**

state **:=** StateCode**.**OK**;**

**if** **not** IsValidTimeSpan**(**spendTime**)** **then** **begin**

state **:=** StateCode**.**INVALID\_SPAN**;**

writeln**(**'invalid'**);**

CreateWorkEntry**.**spendTime **:=** context**.**minimumSpan**;**

**end**

**else** **begin**

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

**end;**

**End;**

{ Adds a WorkEntry to the backed person or adds a new person if person not found }

**Function** AddWorkEntry**(**name**:** **String;** entry**:** WorkEntryRec**):** StateCode**;**

**Var**

node**:** PersonEntry**;**

workEntries**:** WorkEntryList**;**

entries**:** WorkEntryList**;**

hash**:** LongInt**;**

state**:** HashState**;**

**Begin**

AddWorkEntry **:=** StateCode**.**OK**;**

// Check for invalid TimeSpan set on WorkEntry

**if** **((**TimeSpanToSeconds**(**entry**.**spendTime**).**error **<>** ''**)** **or** **(**TimeSpanUnit**.**TimeSpanToSeconds**(**entry**.**spendTime**).**timeInSeconds **<** TimeSpanUnit**.**TimeSpanToSeconds**(**context**.**minimumSpan**).**timeInSeconds**)** **or** **(**TimeSpanUnit**.**TimeSpanToSeconds**(**entry**.**spendTime**).**timeInSeconds **>** TimeSpanUnit**.**TimeSpanToSeconds**(**context**.**maximumSpan**).**timeInSeconds**))then** **begin**

AddWorkEntry **:=** StateCode**.**INVALID\_SPAN**;**

**end**

**else** **begin**

hash **:=** HashUtils**.**ComputeHashOrdinalOnly**(**name**,** Low**(**storage**),** High**(**storage**),** state**);**

// No error on hash calculation

**if** **(**state **=** HashState**.**OK**)** **then** **begin**

// Add entry to person

entries **:=** CreateWorkEntryNode**(**entry**);**

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

// New person

**if** **(**node **=** **nil)** **then** **begin**

node **:=** CreatePersonEntry**(**name**,** entries**);**

// Check for storage overflow

**if** **((**size **+** 1**)** **<** context**.**storageSize**)** **then** **begin**

// Add to chain if element already reside on this index

**if** **(**storage**[**hash**]** **<>** **nil)** **then** **begin**

node**^.**next **:=** storage**[**hash**];**

{ writeln('Added to chain [hash=', hash, ']: ', node^.name); }

**end**

**else** **begin**

{ writeln('Added new index [hash=', hash, ']: ', node^.name); }

**end;**

storage**[**hash**]** **:=** node**;**

size **:=** size **+** 1**;**

**end**

// Overflows storage size

**else** **begin**

AddWorkEntry **:=** StateCode**.**STORAGE\_FULL**;**

**end;**

**end**

// Add WorkEntry

**else** **begin**

workEntries **:=** node**^.**entries**;**

node**^.**entries **:=** entries**;**

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

**end;**

**end;**

**end;**

**end;**

{ Gets the total work time for the given person }

**Procedure** GetTotalWorkTimeForPerson**(**name**:** **String;** **Var** span**:** TimeSpan**;** **Var** state**:** StateCode**);**

**Var**

person**:** PersonEntry**;**

entry**:** WorkEntryList**;**

sec**:** LONGINT**;**

**Begin**

state **:=** StateCode**.**OK**;**

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

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

// Only get total time when person were found

**if** **(**person **<>** **nil)** **then** **begin**

entry **:=** person**^.**entries**;**

**while** **(**entry **<>** **nil)** **do** **begin**

sec **:=** sec **+** TimeSpanUnit**.**TimeSpanToSeconds**(**entry**^.**workEntry**.**spendTime**).**timeInSeconds**;**

entry **:=** entry**^.**next**;**

**end;**

**end**

**else** **begin**

state **:=** StateCode**.**PERSON\_NOT\_FOUND**;**

**end;**

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

**End;**

{ Removes person from the storage }

**Function** RemovePerson**(**name**:** **String):** StateCode**;**

**Var**

pred**,** succ**:** PersonEntry**;**

lowerName**:** **String;**

hash**:** Longint**;**

state**:** HashState**;**

**Begin**

RemovePerson **:=** StateCode**.**OK**;**

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

hash **:=** HashUtils**.**ComputeHashOrdinalOnly**(**lowerName**,** Low**(**storage**),** High**(**storage**),** state**);**

**if** **(**state **<>** HashState**.**OK**)** **then** **begin**

RemovePerson **:=** StateCode**.**PERSON\_NOT\_FOUND**;**

**end**

**else** **begin**

succ **:=** storage**[**hash**];**

pred **:=** **nil;**

writeln**(**succ **<>** **nil);**

**while** **(**succ **<>** **nil)** **and** **(**succ**^.**name **<>** lowerName**)** **do** **begin**

pred **:=** succ**;**

succ **:=** pred**^.**next**;**

**end;**

**if** **(**succ **<>** **nil)** **then** **begin**

**if** **(**pred **<>** **nil)** **then** **begin**

pred**^.**next **:=** succ**^.**next**;**

**end**

**else** **begin**

storage**[**hash**]** **:=** succ**^.**next**;**

**end;**

RemovePersonEntry**(**succ**);**

size **:=** size **-** 1**;**

**end**

**else** **begin**

RemovePerson **:=** StateCode**.**PERSON\_NOT\_FOUND**;**

**end;**

**end;**

**End;**

// For testing, must be removed when used productive

// ################ For testing ################

**procedure** PrintPersons**;**

**Var**

person**:** PersonEntry**;**

entry**:** WorkEntryList**;**

i**:** LongInt**;**

**Begin**

PrintHandler**.**PrintHeader**(**'Storage content'**);**

**for** i **:=** Low**(**storage**)** **to** High**(**Storage**)** **do** **begin**

person **:=** storage**[**i**];**

**if** **(**person **<>** **nil)** **then** **begin**

writeln**(**'hash: '**,** i**);**

**while** **(**person **<>** **nil)** **do** **begin**

write**(**'name: '**,** person**^.**name**,** ': '**);**

entry **:=** person**^.**entries**;**

**while** **(**entry **<>** **nil)** **do** **begin**

write**(**TimeSpanUnit**.**TimeSpanToString**(**entry**^.**workEntry**.**spendTime**),** ', '**);**

entry **:=** entry**^.**next**;**

**end;**

writeln**;**

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

**end;**

writeln**;**

**end;**

**end;**

**End;**

// ################ For testing ################

{ Initialize module }

**Begin**

PrintHandler**.**PrintHeader**(**'Initializing WorkManagementListUnit'**);**

SetLength**(**storage**,** 1**);**

size **:=** 0**;**

writeln**(**'storage is empty'**);**

writeln**(**'size = 0'**);**

writeln**(**'Creating default context...'**);**

writeln**;**

SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** DEFAULT\_STORAGE\_SIZE**,** state**);**

// Stop when default context cannot be initialized

**if** **(**state **<>** StateCode**.**OK**)** **then** **begin**

PrintHandler**.**PrintError**(**'Invalid context detected, program will exit'**);**

Halt**;**

**end;**

**End.**

## WorkManangementHashChainedUnitTest

Folgend ist der Source der WorkManangementHashChainedUnitTest angeführt, der die WorkManangementHashChainedUnit testet.

**Program** WorkManagementHashChainedUnitTest**;**

**Uses** sysutils**,** Crt**,** WorkManagementHashChainedUnit**,** PrintUtils**,** TimeSpanUnit**;**

**Var**

state**:** StateCode**;**

{

Test the procedure which allows setting of the context.

}

**Procedure** TestSetWorkEntryContext**;**

**Begin**

// Invalid MinimumSpan

PrintUtils**.**PrintHeader**(**'Invalid MinimumSpan'**);**

WorkManagementHashChainedUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 60**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**),** 10**,** state**);**

writeln**;**

// Invalid MaximumSpan

PrintUtils**.**PrintHeader**(**'Invalid MaximumSpan'**);**

WorkManagementHashChainedUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 1**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 60**),** 10**,** state**);**

writeln**;**

// Invalid Range

PrintUtils**.**PrintHeader**(**'Invalid Range'**);**

WorkManagementHashChainedUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 30**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 10**),** 10**,** state**);**

writeln**;**

// Invalid StorageSize

PrintUtils**.**PrintHeader**(**'Invalid Storage size'**);**

WorkManagementHashChainedUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 30**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 50**),** 0**,** state**);**

writeln**;**

// Valid context

PrintUtils**.**PrintHeader**(**'Valid context definition'**);**

WorkManagementHashChainedUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 30**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 50**),** 10**,** state**);**

**End;**

{

Tests the function CreateWorkEntry

}

**Procedure** TestCreateWorkEntry**;**

**Var**

entry**:** WorkEntryRec**;**

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

state**:** StateCode**;**

**Begin**

WorkManagementHashChainedUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 100**,** state**);**

// Invalid TimeSpan

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 60**);**

entry **:=** WorkManagementHashChainedUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'Invalid TimeSpan'**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'WorkEntry.spendTime: '**,** TimeSpanUnit**.**TimeSpanToString**(**entry**.**spendTime**));**

writeln**(**'State : '**,** state**);**

writeln**;**

// TimeSpan overflows minimum

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 59**);**

entry **:=** WorkManagementHashChainedUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'TimeSpan overflow minimum'**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'WorkEntry.spendTime: '**,** TimeSpanUnit**.**TimeSpanToString**(**entry**.**spendTime**));**

writeln**(**'State : '**,** state**);**

// TimeSpan overflows maximum

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 1**);**

entry **:=** WorkManagementHashChainedUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'TimeSpan overflow maximum'**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'WorkEntry.spendTime: '**,** TimeSpanUnit**.**TimeSpanToString**(**entry**.**spendTime**));**

writeln**(**'State : '**,** state**);**

// Valid TimeSpan

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**);**

entry **:=** WorkManagementHashChainedUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'Valid TimeSpan'**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'WorkEntry.spendTime: '**,** TimeSpanUnit**.**TimeSpanToString**(**entry**.**spendTime**));**

writeln**(**'State : '**,** state**);**

**End;**

{

Tests the function AddWorkEntry

}

**Procedure** TestAddWorkEntry**;**

**Var**

entry**:** WorkEntryRec**;**

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

state**:** StateCode**;**

i**,** j**:** Integer**;**

**Begin**

// Invalid Time span

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 60**);**

entry**.**spendTime **:=** span**;**

PrintUtils**.**PrintHeader**(**'Invalid TimeSpan '**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** WorkManagementHashChainedUnit**.**AddWorkEntry**(**'thomas'**,** entry**));**

writeln**;**

// Overlfows minimum span

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 1**);**

entry**.**spendTime **:=** span**;**

PrintUtils**.**PrintHeader**(**'Overflows minimumSpan '**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** WorkManagementHashChainedUnit**.**AddWorkEntry**(**'thomas'**,** entry**));**

writeln**;**

// Overlfows maximum span

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 1**);**

entry**.**spendTime **:=** span**;**

PrintUtils**.**PrintHeader**(**'Overflows maximumSpan '**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** WorkManagementHashChainedUnit**.**AddWorkEntry**(**'thomas'**,** entry**));**

writeln**;**

// Overlfows maximum storage

WorkManagementHashChainedUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 6**,** state**);**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**);**

entry **:=** WorkManagementHashChainedUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'Overflows storage size '**);**

**for** i **:=** 1 **to** 3 **do** **begin**

writeln**(**'Try to add : '**,** 'thomas\_'**,** i**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

state **:=** WorkManagementHashChainedUnit**.**AddWorkEntry**(**'thomas\_' **+** IntToStr**(**i**),** entry**);**

writeln**(**'State : '**,** state**);**

writeln**(**'Try to add : '**,** 'wolfganf\_'**,** i**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

state **:=** WorkManagementHashChainedUnit**.**AddWorkEntry**(**'wolfganf\_' **+** IntToStr**(**i**),** entry**);**

writeln**(**'State : '**,** state**);**

writeln**;**

**end;**

writeln**;**

PrintPersons**;**

// Normal behavior

WorkManagementHashChainedUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 3**,** state**);**

PrintUtils**.**PrintHeader**(**'Normal behavior '**);**

**for** i **:=** 1 **to** 3 **do** **begin**

writeln**(**'Try to add : '**,** 'thomas\_'**,** i**);**

**For** j **:=** 1 **to** 5 **do** **begin**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** j**);**

entry **:=** WorkManagementHashChainedUnit**.**CreateWorkEntry**(**span**,** state**);**

state **:=** WorkManagementHashChainedUnit**.**AddWorkEntry**(**'thomas\_' **+** IntToStr**(**i**),** entry**);**

**end;**

writeln**(**'State : '**,** state**);**

writeln**;**

**end;**

writeln**;**

PrintPersons**;**

**End;**

{

Tests the procedure GetTotalWorkTimeForPerson

}

**Procedure** TestGetTotalWorkTimeForPerson**;**

**Var**

entry**:** WorkEntryRec**;**

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

state**:** StateCode**;**

i**,** j**:** Integer**;**

**Begin**

WorkManagementHashChainedUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 10**,** state**);**

**for** i **:=** 1 **to** 11 **do** **begin**

**For** j **:=** 1 **to** 5 **do** **begin**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** j**);**

entry **:=** WorkManagementHashChainedUnit**.**CreateWorkEntry**(**span**,** state**);**

state **:=** WorkManagementHashChainedUnit**.**AddWorkEntry**(**'thomas\_' **+** IntToStr**(**i**),** entry**);**

**end;**

**end;**

// Person not found

PrintUtils**.**PrintHeader**(**'Person not found'**);**

WorkManagementHashChainedUnit**.**GetTotalWorkTimeForPerson**(**'thomas\_11'**,** span**,** state**);**

writeln**(**'Person: '**,** 'Not found'**);**

writeln**(**'Span : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** state**);**

writeln**;**

// Normal bhavor

PrintUtils**.**PrintHeader**(**'Person found'**);**

WorkManagementHashChainedUnit**.**GetTotalWorkTimeForPerson**(**'thomas\_10'**,** span**,** state**);**

writeln**(**'Person: '**,** 'thomas\_10'**);**

writeln**(**'Span : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** state**);**

**End;**

{

Test the function RemovePerson

}

**procedure** TestRemovePerson**;**

**Var**

entry**:** WorkEntryRec**;**

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

state**:** StateCode**;**

i**,** j**:** Integer**;**

**Begin**

WorkManagementHashChainedUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 10**,** state**);**

**for** i **:=** 1 **to** 5 **do** **begin**

**For** j **:=** 1 **to** 5 **do** **begin**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** j**);**

entry **:=** WorkManagementHashChainedUnit**.**CreateWorkEntry**(**span**,** state**);**

state **:=** WorkManagementHashChainedUnit**.**AddWorkEntry**(**'thomas\_' **+** IntToStr**(**i**),** entry**);**

state **:=** WorkManagementHashChainedUnit**.**AddWorkEntry**(**'wolfganf\_' **+** IntToStr**(**i**),** entry**);**

**end;**

**end;**

// Person not found

PrintUtils**.**PrintHeader**(**'Person not found in storage'**);**

state **:=** WorkManagementHashChainedUnit**.**RemovePerson**(**'thomas\_6'**);**

writeln**(**'Person: '**,** 'thomas\_6'**);**

writeln**(**'State : '**,** state**);**

WorkManagementHashChainedUnit**.**PrintPersons**;**

writeln**;**

// First Person removed

PrintUtils**.**PrintHeader**(**'First Person removed'**);**

state **:=** WorkManagementHashChainedUnit**.**RemovePerson**(**'thomas\_1'**);**

writeln**(**'Person: '**,** 'thomas\_1'**);**

writeln**(**'State : '**,** state**);**

WorkManagementHashChainedUnit**.**PrintPersons**;**

writeln**;**

// Remove from chain

PrintUtils**.**PrintHeader**(**'Remove from chain'**);**

WorkManagementHashChainedUnit**.**PrintPersons**;**

state **:=** WorkManagementHashChainedUnit**.**RemovePerson**(**'thomas\_5'**);**

writeln**(**'Person: '**,** 'thomas\_5'**);**

writeln**(**'State : '**,** state**);**

state **:=** WorkManagementHashChainedUnit**.**RemovePerson**(**'wolfganf\_3'**);**

writeln**(**'Person: '**,** 'wolfganf\_3'**);**

writeln**(**'State : '**,** state**);**

WorkManagementHashChainedUnit**.**PrintPersons**;**

writeln**;**

**End;**

**Begin**

// Test SetWorkEntryContext

PrintUtils**.**Print**(**'Tests for procedure SetWorkEntryContext'**,** Green**,** White**);**

TestSetWorkEntryContext**;**

WorkManagementHashChainedUnit**.**CleanStorage**;**

writeln**;**

// Test CreateWorkEntry

PrintUtils**.**Print**(**'Tests for procedure CreateWorkEntry'**,** Green**,** White**);**

TestCreateWorkEntry**;**

WorkManagementHashChainedUnit**.**CleanStorage**;**

writeln**;**

// Test AddWorkEntry

PrintUtils**.**Print**(**'Tests for procedure AddWorkEntry'**,** Green**,** White**);**

TestAddWorkEntry**;**

WorkManagementHashChainedUnit**.**CleanStorage**;**

writeln**;**

// Test GetTotalWorkTimeForPerson

PrintUtils**.**Print**(**'Tests for procedure GetTotalWorkTimeForPerson'**,** Green**,** White**);**

TestGetTotalWorkTimeForPerson**;**

WorkManagementHashChainedUnit**.**CleanStorage**;**

writeln**;**

// Test RemovePerson

PrintUtils**.**Print**(**'Tests for procedure RemovePerson'**,** Green**,** White**);**

TestRemovePerson**;**

WorkManagementHashChainedUnit**.**CleanStorage**;**

writeln**;**

**End.**

## WorkManagementHashOpenUnit

Folgend ist der Source der WorkManagementHashOpenUnit angeführt, welche ein Hashing Verfahren ohne zusätzliche Datenstruktur verwendet und eine offene Adressierung für die Kollisionsbehandlung verwendet.

{

This implementation implements the interface WorkManagementInterface and uses

a Single-Connected-Linked-list as the storage.

The default storage size is set to: 10.

The default minimum TimeSpan is set to: 0:1:0

The default maximum TimeSpan is set to: 8:0:0

}

**Unit** WorkManagementHashOpenUnit**;**

{$I WorkManagementInterface.pas}

// For testing, must be removed when used productive

// ################ For testing ################

**procedure** PrintPersons**;**

// ################ For testing ################

**Implementation**

**Uses**

HashUtils**;**

**Type**

// Pointer to the WorkEntryNodeRec

WorkEntryNode **=** **^**WorkEntry**;**

// Pointer to the WorkEntryNode used as separate tpye for list

WorkEntryList **=** WorkEntryNode**;**

// WorkEntry which hold WorkEntryRec which gets used for list node

WorkEntry **=** **Record**

workEntry**:** WorkEntryRec**;**

next**:** WorkEntryNode**;**

**End;**

// Pointer to the PersonEntryRec type

PersonEntry **=** **^**PersonEntryRec**;**

// The person entry record which holds a work entry

PersonEntryRec **=** **Record**

name**:** **String;**

entries**:** WorkEntryList**;**

deleted**:** Boolean**;**

**end;**

{

Unit visible variables

}

**Var**

// Single connected list used as storage

storage**:** **Array** **of** PersonEntry**;**

// The context of this module

context**:** WorkEntryContext**;**

// Internal State holder

state**:** StateCode**;**

// current size of the list

size**:** Longint**;**

**Const**

DEFAULT\_STORAGE\_SIZE **=** 1**;**

// ################# Private function and procedure #################

{

Creates a person entry instance. Be aware that this method returns a pointer and you will need to

dispose this instance manually.

@param

name: the name of the person

@param

entries: the WorkEntryList instance which will be set on the person entry

@return

the created PersonEntry instance

}

**Function** CreatePersonEntry**(**name**:** **String;** entries**:** WorkEntryList**):** PersonEntry**;**

**Begin**

// Create PersonEntry

CreatePersonEntry **:=** New**(**PersonEntry**);**

CreatePersonEntry**^.**name **:=** LowerCase**(**name**);**

CreatePersonEntry**^.**entries **:=** entries**;**

CreatePersonEntry**^.**deleted **:=** false**;**

**End;**

{

Creates a WorkEntryNode for the given workEntryRec

@param

entry: the WorkEntryRec to create WorkEntryNode

@return

the created WorkEntryNode

}

**Function** CreateWorkEntryNode**(**entry**:** WorkEntryRec**):** WorkEntryNode**;**

**Begin**

CreateWorkEntryNode **:=** New**(**WorkEntryNode**);**

CreateWorkEntryNode**^.**workEntry **:=** entry**;**

CreateWorkEntryNode**^.**next **:=** **nil;**

**End;**

{

Gets the PersonEntry by its name from the storage.

@param

name: the name of the person to be searched

@return

the found PersonEntry, níll otherwise

}

**Function** GetPersonEntry**(**name**:** **String):** PersonEntry**;**

**Var**

node**:** PersonEntry**;**

lowerName**:** **String;**

hash**,** i**:** LongInt**;**

state**:** HashState**;**

**Begin**

node **:=** **nil;**

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

hash **:=** HashUtils**.**computeHashOrdinalOnly**(**lowerName**,** Low**(**storage**),** High**(**storage**),** state**);**

**if** **(**state **=** HashState**.**OK**)** **then** **begin**

node **:=** storage**[**hash**];**

// Wrong person found on coputed index

**if** **(**node **<>** **nil)** **and** **(**node**^.**name **<>** lowerName**)** **then** **begin**

// check for already last index

**if** **(**hash **=** High**(**storage**))** **then** **begin**

i **:=** Low**(**storage**);**

**end**

**else** **begin**

i **:=** hash **+** 1**;**

**end;**

// search on other indexes

**while** **(**storage**[**i**]** **<>** **nil)** **and** **(**i **<>** hash**)** **and** **(**storage**[**i**]^.**name **<>** lowerName**)** **do** **begin**

Inc**(**i**);**

**if** **(**i **>** High**(**storage**))** **then** **begin**

i **:=** Low**(**storage**);**

**end;**

**end;**

// If not same index, not nil person then we found it

**if** **(**i **<>** hash**)** **and** **(**storage**[**i**]** **<>** **nil)** **then** **begin**

node **:=** storage**[**i**];**

**end**

// Else we did not found it

**else** **begin**

node **:=** **nil;**

**end;**

**end;**

**end;**

GetPersonEntry **:=** node**;**

**End;**

{

Removes a person by deleting all of its contained WorkEntryNodes

and the person itself. The caller must ensure that the next component

is properly connected to the previous of this person, otherwise

the list will be broken.

}

**Procedure** RemovePersonEntry**(**person**:** PersonEntry**);**

**Var**

pred**,** succ**:** WorkEntryList**;**

**Begin**

**if** **(**person **<>** **nil)** **then** **begin**

pred **:=** person**^.**entries**;**

succ **:=** **nil;**

**while** **(**pred **<>** **nil)** **do** **begin**

succ **:=** pred**^.**next**;**

Dispose**(**pred**);**

{ WriteLn('WorkEntry disposed'); }

pred **:=** succ**;**

**end;**

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

{ WriteLn('PersonEntry disposed'); }

size **:=** size **-** 1**;**

**end;**

**End;**

{

Validates if the given TimeSpan is within context borders.

@param

span: the TimeSpan instance to validate

@return

true if the given TimeSpan instance is within borders, false otherwise

}

**Function** IsValidTimeSpan**(**spendTime**:** TimeSpan**):** Boolean**;**

**Begin**

IsValidTImeSpan **:=** **((**TimeSpanToSeconds**(**spendTime**).**error **=** ''**)** **and** **(**TimeSpanUnit**.**TimeSpanToSeconds**(**spendTime**).**timeInSeconds **>=** TimeSpanUnit**.**TimeSpanToSeconds**(**context**.**minimumSpan**).**timeInSeconds**)** **and** **(**TimeSpanUnit**.**TimeSpanToSeconds**(**spendTime**).**timeInSeconds **<=** TimeSpanUnit**.**TimeSpanToSeconds**(**context**.**maximumSpan**).**timeInSeconds**))**

**End;**

// ################# Interface function and procedure #################

**Procedure** SetWorkEntryContext**(**minimumSpan**,** maximumSpan**:** TimeSpan**;** storageSize**:** Longint**;** **Var** state**:** StateCode**);**

**Begin**

state **:=** StateCode**.**OK**;**

// set error if context definition is invalid

**if** **((**TimeSpanUnit**.**TimeSpanToSeconds**(**minimumSpan**).**error **<>** ''**)** **or** **(**TimeSpanUnit**.**TimeSpanToSeconds**(**maximumSpan**).**error **<>** ''**)** **or** **(**TimeSpanUnit**.**IsShorterThan**(**minimumSpan**,** maximumSpan**)** **=** false**)** **or** **(**storageSize **<** DEFAULT\_STORAGE\_SIZE**))** **then** **begin**

state **:=** StateCode**.**INVALID\_CONTEXT**;**

PrintHandler**.**PrintError**(**'Context invalid'**);**

**end**

// set context if valid definition

**else** **begin**

context**.**minimumSpan **:=** minimumSpan**;**

context**.**maximumSpan **:=** maximumSpan**;**

context**.**storageSize **:=** storageSize **+** 1**;**

CleanStorage**;**

SetLength**(**storage**,** context**.**storageSize**);**

PrintHandler**.**PrintHeader**(**'Context successfully set'**);**

**end;**

writeln**(**'minimumSpan: '**,** TimeSpanUnit**.**TimeSpanToString**(**minimumSpan**));**

writeln**(**'maximumSpan:'**,** TimeSpanUnit**.**TimeSpanToString**(**maximumSpan**));**

writeln**(**'storageSize: '**,** storageSize**);**

writeln**;**

**End;**

{ Cleans the storage }

**Procedure** CleanStorage**;**

**Var**

person**:** PersonEntry**;**

i**:** LongInt**;**

**Begin**

**for** i **:=** Low**(**storage**)** **to** High**(**storage**)** **do** **begin**

person **:=** storage**[**i**];**

RemovePersonEntry**(**person**);**

storage**[**i**]** **:=** **nil;**

**end;**

size **:=** 0**;**

**End;**

{ Creates a WorkEntry instance }

{ Creates a WorkEntry instance }

**Function** CreateWorkEntry**(**spendTime**:** TimeSpan**;** **VAR** state**:** StateCode**):** WorkEntryRec**;**

**Begin**

state **:=** StateCode**.**OK**;**

**if** **not** IsValidTimeSpan**(**spendTime**)** **then** **begin**

state **:=** StateCode**.**INVALID\_SPAN**;**

writeln**(**'invalid'**);**

CreateWorkEntry**.**spendTime **:=** context**.**minimumSpan**;**

**end**

**else** **begin**

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

**end;**

**End;**

{ Adds a WorkEntry to the backed person or adds a new person if person not found }

**Function** AddWorkEntry**(**name**:** **String;** entry**:** WorkEntryRec**):** StateCode**;**

**Var**

node**:** PersonEntry**;**

workEntries**:** WorkEntryList**;**

entries**:** WorkEntryList**;**

hash**,** i**:** LongInt**;**

state**:** HashState**;**

workState**:** StateCode**;**

**Begin**

workState **:=** StateCode**.**OK**;**

// Check for invalid TimeSpan set on WorkEntry

**if** **((**TimeSpanToSeconds**(**entry**.**spendTime**).**error **<>** ''**)** **or** **(**TimeSpanUnit**.**TimeSpanToSeconds**(**entry**.**spendTime**).**timeInSeconds **<** TimeSpanUnit**.**TimeSpanToSeconds**(**context**.**minimumSpan**).**timeInSeconds**)** **or** **(**TimeSpanUnit**.**TimeSpanToSeconds**(**entry**.**spendTime**).**timeInSeconds **>** TimeSpanUnit**.**TimeSpanToSeconds**(**context**.**maximumSpan**).**timeInSeconds**))then** **begin**

workState **:=** StateCode**.**INVALID\_SPAN**;**

**end**

**else** **begin**

hash **:=** HashUtils**.**ComputeHashOrdinalOnly**(**name**,** Low**(**storage**),** High**(**storage**),** state**);**

i **:=** hash**;**

// No error on hash calculation

**if** **(**state **=** HashState**.**OK**)** **then** **begin**

// Add entry to person

entries **:=** CreateWorkEntryNode**(**entry**);**

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

// New person to be added

**if** **(**node **=** **nil)** **then** **begin**

node **:=** CreatePersonEntry**(**name**,** entries**);**

// Check for storage overflow

**if** **((**size **+** 1**)** **<** context**.**storageSize**)** **then** **begin**

// Check if computed hash index is already in use

**if** **(**storage**[**hash**]** **<>** **nil)** **then** **begin**

// check for already last index

**if** **(**hash **=** High**(**storage**))** **then** **begin**

i **:=** Low**(**storage**);**

**end**

**else** **begin**

i **:=** hash **+** 1**;**

**end;**

// search for free indexes

**while** **(**storage**[**i**]** **<>** **nil)** **and** **(**i **<>** hash**)** **and** **(not** storage**[**i**]^.**deleted**)** **do** **begin**

Inc**(**i**);**

**if** **(**i **>** High**(**storage**))** **then** **begin**

i **:=** Low**(**storage**);**

**end;**

**end;**

// Else no place left

// Should never occur regarding kept storage element count

**if** **(**i **=** hash**)** **then** **begin**

workState **:=** StateCode**.**STORAGE\_FULL**;**

**end**

**end;**

**end**

// Overflows storage size

**else** **begin**

workState **:=** StateCode**.**STORAGE\_FULL**;**

**end;**

// Free to add person

**if** **(**workState **=** StateCode**.**OK**)** **then** **begin**

// If a deleted person reside then remove it

**if** **(**storage**[**i**]** **<>** **nil)** **then** **begin**

RemovePersonEntry**(**storage**[**i**]);**

**end;**

storage**[**i**]** **:=** node**;**

writeln**(**'hash['**,** hash**,** ']-idx['**,** i**,** ']: '**,** node**^.**name**);**

size **:=** size **+** 1**;**

**end**

**end**

// Add WorkEntry

**else** **begin**

workEntries **:=** node**^.**entries**;**

node**^.**entries **:=** entries**;**

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

**end;**

**end;**

**end;**

AddWorkEntry **:=** workState**;**

**end;**

{ Gets the total work time for the given person }

**Procedure** GetTotalWorkTimeForPerson**(**name**:** **String;** **Var** span**:** TimeSpan**;** **Var** state**:** StateCode**);**

**Var**

person**:** PersonEntry**;**

entry**:** WorkEntryList**;**

sec**:** LONGINT**;**

**Begin**

state **:=** StateCode**.**OK**;**

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

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

// Only get total time when person were found

**if** **(**person **<>** **nil)** **then** **begin**

entry **:=** person**^.**entries**;**

**while** **(**entry **<>** **nil)** **do** **begin**

sec **:=** sec **+** TimeSpanUnit**.**TimeSpanToSeconds**(**entry**^.**workEntry**.**spendTime**).**timeInSeconds**;**

entry **:=** entry**^.**next**;**

**end;**

**end**

**else** **begin**

state **:=** StateCode**.**PERSON\_NOT\_FOUND**;**

**end;**

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

**End;**

{ Removes person from the storage }

**Function** RemovePerson**(**name**:** **String):** StateCode**;**

**Var**

person**:** PersonEntry**;**

state**:** HashState**;**

**Begin**

RemovePerson **:=** StateCode**.**OK**;**

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

// Mark as deleted

**if** **(**person **<>** **nil)** **and** **(not** person**^.**deleted**)** **then** **begin**

person**^.**deleted **:=** true**;**

size **:=** size **-** 1**;**

**end**

**else** **begin**

RemovePerson **:=** StateCode**.**PERSON\_NOT\_FOUND**;**

**end;**

**End;**

// For testing, must be removed when used productive

// ################ For testing ################

**procedure** PrintPersons**;**

**Var**

person**:** PersonEntry**;**

entry**:** WorkEntryList**;**

i**:** LongInt**;**

**Begin**

PrintHandler**.**PrintHeader**(**'Storage content'**);**

**for** i **:=** Low**(**storage**)** **to** High**(**Storage**)** **do** **begin**

person **:=** storage**[**i**];**

**if** **(**person **<>** **nil)** **then** **begin**

write**(**'hash['**,** i**,** ']-deleted['**,** person**^.**deleted**,** ']: '**,** person**^.**name**,** ': '**);**

entry **:=** person**^.**entries**;**

**while** **(**entry **<>** **nil)** **do** **begin**

write**(**TimeSpanUnit**.**TimeSpanToString**(**entry**^.**workEntry**.**spendTime**),** ', '**);**

entry **:=** entry**^.**next**;**

**end;**

writeln**;**

**end;**

**end;**

**End;**

// ################ For testing ################

{ Initialize module }

**Begin**

PrintHandler**.**PrintHeader**(**'Initializing WorkManagementListUnit'**);**

SetLength**(**storage**,** 1**);**

size **:=** 0**;**

writeln**(**'storage is empty'**);**

writeln**(**'size = 0'**);**

writeln**(**'Creating default context...'**);**

writeln**;**

SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** DEFAULT\_STORAGE\_SIZE**,** state**);**

// Stop when default context cannot be initialized

**if** **(**state **<>** StateCode**.**OK**)** **then** **begin**

PrintHandler**.**PrintError**(**'Invalid context detected, program will exit'**);**

Halt**;**

**end;**

**End.**

## WorkManagementHashOpenUnitTest

Folgend ist der Source der WorkManagementHashUnitOpenTest angeführt welche die Unit WorkManagementHashOpenUnitTest testet.

**Program** WorkManagementHashOpenUnitTest**;**

**Uses** sysutils**,** Crt**,** WorkManagementHashOpenUnit**,** PrintUtils**,** TimeSpanUnit**;**

**Var**

state**:** StateCode**;**

{

Test the procedure which allows setting of the context.

}

**Procedure** TestSetWorkEntryContext**;**

**Begin**

// Invalid MinimumSpan

PrintUtils**.**PrintHeader**(**'Invalid MinimumSpan'**);**

WorkManagementHashOpenUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 60**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 2**,** 0**),** 10**,** state**);**

writeln**;**

// Invalid MaximumSpan

PrintUtils**.**PrintHeader**(**'Invalid MaximumSpan'**);**

WorkManagementHashOpenUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 1**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 60**),** 10**,** state**);**

writeln**;**

// Invalid Range

PrintUtils**.**PrintHeader**(**'Invalid Range'**);**

WorkManagementHashOpenUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 30**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 10**),** 10**,** state**);**

writeln**;**

// Invalid StorageSize

PrintUtils**.**PrintHeader**(**'Invalid Storage size'**);**

WorkManagementHashOpenUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 30**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 50**),** 0**,** state**);**

writeln**;**

// Valid context

PrintUtils**.**PrintHeader**(**'Valid context definition'**);**

WorkManagementHashOpenUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 30**),** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 50**),** 10**,** state**);**

**End;**

{

Tests the function CreateWorkEntry

}

**Procedure** TestCreateWorkEntry**;**

**Var**

entry**:** WorkEntryRec**;**

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

state**:** StateCode**;**

**Begin**

WorkManagementHashOpenUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 100**,** state**);**

// Invalid TimeSpan

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 60**);**

entry **:=** WorkManagementHashOpenUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'Invalid TimeSpan'**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'WorkEntry.spendTime: '**,** TimeSpanUnit**.**TimeSpanToString**(**entry**.**spendTime**));**

writeln**(**'State : '**,** state**);**

writeln**;**

// TimeSpan overflows minimum

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 59**);**

entry **:=** WorkManagementHashOpenUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'TimeSpan overflow minimum'**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'WorkEntry.spendTime: '**,** TimeSpanUnit**.**TimeSpanToString**(**entry**.**spendTime**));**

writeln**(**'State : '**,** state**);**

// TimeSpan overflows maximum

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 1**);**

entry **:=** WorkManagementHashOpenUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'TimeSpan overflow maximum'**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'WorkEntry.spendTime: '**,** TimeSpanUnit**.**TimeSpanToString**(**entry**.**spendTime**));**

writeln**(**'State : '**,** state**);**

// Valid TimeSpan

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**);**

entry **:=** WorkManagementHashOpenUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'Valid TimeSpan'**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'WorkEntry.spendTime: '**,** TimeSpanUnit**.**TimeSpanToString**(**entry**.**spendTime**));**

writeln**(**'State : '**,** state**);**

**End;**

{

Tests the function AddWorkEntry

}

**Procedure** TestAddWorkEntry**;**

**Var**

entry**:** WorkEntryRec**;**

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

state**:** StateCode**;**

i**,** j**:** Integer**;**

**Begin**

// Invalid Time span

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 60**);**

entry**.**spendTime **:=** span**;**

PrintUtils**.**PrintHeader**(**'Invalid TimeSpan '**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** WorkManagementHashOpenUnit**.**AddWorkEntry**(**'thomas'**,** entry**));**

writeln**;**

// Overlfows minimum span

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 0**,** 1**);**

entry**.**spendTime **:=** span**;**

PrintUtils**.**PrintHeader**(**'Overflows minimumSpan '**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** WorkManagementHashOpenUnit**.**AddWorkEntry**(**'thomas'**,** entry**));**

writeln**;**

// Overlfows maximum span

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 1**);**

entry**.**spendTime **:=** span**;**

PrintUtils**.**PrintHeader**(**'Overflows maximumSpan '**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** WorkManagementHashOpenUnit**.**AddWorkEntry**(**'thomas'**,** entry**));**

writeln**;**

// Overlfows maximum storage

WorkManagementHashOpenUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 4**,** state**);**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**);**

entry **:=** WorkManagementHashOpenUnit**.**CreateWorkEntry**(**span**,** state**);**

PrintUtils**.**PrintHeader**(**'Overflows storage size '**);**

**for** i **:=** 1 **to** 3 **do** **begin**

writeln**(**'Try to add : '**,** 'thomas\_'**,** i**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

state **:=** WorkManagementHashOpenUnit**.**AddWorkEntry**(**'thomas\_' **+** IntToStr**(**i**),** entry**);**

writeln**(**'State : '**,** state**);**

writeln**(**'-------------------------------------'**);**

writeln**(**'Try to add : '**,** 'wolfganf\_'**,** i**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

state **:=** WorkManagementHashOpenUnit**.**AddWorkEntry**(**'wolfganf\_' **+** IntToStr**(**i**),** entry**);**

writeln**(**'State : '**,** state**);**

writeln**(**'-------------------------------------'**);**

**end;**

writeln**;**

PrintPersons**;**

// Normal behavior

WorkManagementHashOpenUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 3**,** state**);**

PrintUtils**.**PrintHeader**(**'Normal behavior '**);**

**for** i **:=** 1 **to** 3 **do** **begin**

writeln**(**'Try to add : '**,** 'thomas\_'**,** i**);**

**For** j **:=** 1 **to** 5 **do** **begin**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** j**);**

entry **:=** WorkManagementHashOpenUnit**.**CreateWorkEntry**(**span**,** state**);**

state **:=** WorkManagementHashOpenUnit**.**AddWorkEntry**(**'thomas\_' **+** IntToStr**(**i**),** entry**);**

writeln**(**'Set spendTime : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

**end;**

writeln**(**'State : '**,** state**);**

writeln**;**

**end;**

writeln**;**

PrintPersons**;**

**End;**

{

Tests the procedure GetTotalWorkTimeForPerson

}

**Procedure** TestGetTotalWorkTimeForPerson**;**

**Var**

entry**:** WorkEntryRec**;**

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

state**:** StateCode**;**

i**,** j**:** Integer**;**

**Begin**

WorkManagementHashOpenUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 10**,** state**);**

**for** i **:=** 1 **to** 11 **do** **begin**

**For** j **:=** 1 **to** 5 **do** **begin**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** j**);**

entry **:=** WorkManagementHashOpenUnit**.**CreateWorkEntry**(**span**,** state**);**

state **:=** WorkManagementHashOpenUnit**.**AddWorkEntry**(**'thomas\_' **+** IntToStr**(**i**),** entry**);**

**end;**

**end;**

// Person not found

PrintUtils**.**PrintHeader**(**'Person not found'**);**

WorkManagementHashOpenUnit**.**GetTotalWorkTimeForPerson**(**'thomas\_11'**,** span**,** state**);**

writeln**(**'Person: '**,** 'Not found'**);**

writeln**(**'Span : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** state**);**

writeln**;**

// Normal bhavor

PrintUtils**.**PrintHeader**(**'Person found'**);**

WorkManagementHashOpenUnit**.**GetTotalWorkTimeForPerson**(**'thomas\_10'**,** span**,** state**);**

writeln**(**'Person: '**,** 'thomas\_10'**);**

writeln**(**'Span : '**,** TimeSpanUnit**.**TimeSpanToString**(**span**));**

writeln**(**'State : '**,** state**);**

**End;**

{

Test the function RemovePerson

}

**procedure** TestRemovePerson**;**

**Var**

entry**:** WorkEntryRec**;**

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

state**:** StateCode**;**

i**,** j**:** Integer**;**

**Begin**

WorkManagementHashOpenUnit**.**SetWorkEntryContext**(**TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** 0**),** TimeSpanUnit**.**CreateTimeSpan**(**8**,** 0**,** 0**),** 10**,** state**);**

**for** i **:=** 1 **to** 5 **do** **begin**

**For** j **:=** 1 **to** 5 **do** **begin**

span **:=** TimeSpanUnit**.**CreateTimeSpan**(**0**,** 1**,** j**);**

entry **:=** WorkManagementHashOpenUnit**.**CreateWorkEntry**(**span**,** state**);**

state **:=** WorkManagementHashOpenUnit**.**AddWorkEntry**(**'thomas\_' **+** IntToStr**(**i**),** entry**);**

state **:=** WorkManagementHashOpenUnit**.**AddWorkEntry**(**'wolfganf\_' **+** IntToStr**(**i**),** entry**);**

**end;**

**end;**

// Person not found

PrintUtils**.**PrintHeader**(**'Person not found in storage'**);**

state **:=** WorkManagementHashOpenUnit**.**RemovePerson**(**'thomas\_6'**);**

writeln**(**'Person: '**,** 'thomas\_6'**);**

writeln**(**'State : '**,** state**);**

WorkManagementHashOpenUnit**.**PrintPersons**;**

writeln**;**

// First Person removed

PrintUtils**.**PrintHeader**(**'Remove persons'**);**

state **:=** WorkManagementHashOpenUnit**.**RemovePerson**(**'thomas\_4'**);**

writeln**(**'Person: '**,** 'thomas\_4'**);**

writeln**(**'State : '**,** state**);**

state **:=** WorkManagementHashOpenUnit**.**RemovePerson**(**'wolfganf\_5'**);**

writeln**(**'Person: '**,** 'wolfganf\_5'**);**

writeln**(**'State : '**,** state**);**

WorkManagementHashOpenUnit**.**PrintPersons**;**

writeln**;**

**End;**

**Begin**

// Test SetWorkEntryContext

PrintUtils**.**Print**(**'Tests for procedure SetWorkEntryContext'**,** Green**,** White**);**

TestSetWorkEntryContext**;**

WorkManagementHashOpenUnit**.**CleanStorage**;**

writeln**;**

// Test CreateWorkEntry

PrintUtils**.**Print**(**'Tests for procedure CreateWorkEntry'**,** Green**,** White**);**

TestCreateWorkEntry**;**

WorkManagementHashOpenUnit**.**CleanStorage**;**

writeln**;**

// Test AddWorkEntry

PrintUtils**.**Print**(**'Tests for procedure AddWorkEntry'**,** Green**,** White**);**

TestAddWorkEntry**;**

WorkManagementHashOpenUnit**.**CleanStorage**;**

writeln**;**

// Test GetTotalWorkTimeForPerson

PrintUtils**.**Print**(**'Tests for procedure GetTotalWorkTimeForPerson'**,** Green**,** White**);**

TestGetTotalWorkTimeForPerson**;**

WorkManagementHashOpenUnit**.**CleanStorage**;**

writeln**;**

// Test RemovePerson

PrintUtils**.**Print**(**'Tests for procedure RemovePerson'**,** Green**,** White**);**

TestRemovePerson**;**

WorkManagementHashOpenUnit**.**CleanStorage**;**

writeln**;**

**End.**

# Tests

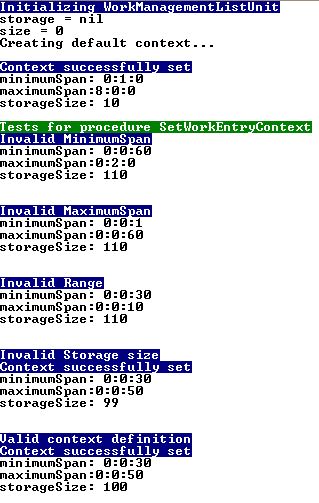
Folgend sind die Tests der Implementierungen angeführt. Auf eine genauere Dokumentation der getestet Units, welche Hilfsfunktionen und Prozeduren enthalten wurde verzichtet. Es sei hierbei auf den Source verwiesen. Hier werden lediglich die implementierten Units angeführt.

## Single-Linked-List Unit

Folgend sind die Tests der Implementierung angeführt, wobei als Storage eine Single-Linked-List verwendet wird.

### SetWorkEntryContext

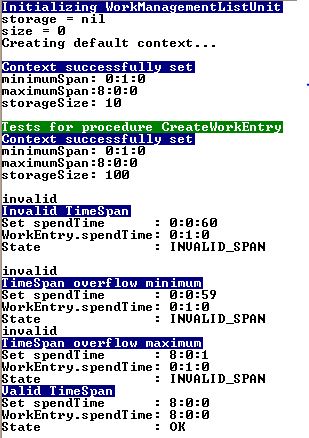
Diese Tests testen die Funktion SetWorklfowEntryContext, welche es erlaubt die Unit zur Laufzeit zu konfigurieren, wobei anzumerken ist, das in diesem Fall auch das Storage gelöscht wird. Solange versucht wird einen ungültigen Kontext zu definieren bleibt der zuletzt gültige aktiv. Beim Initialisieren der Unit wird ein Standardkontext gesetzt, da es immer einen aktiven Kontext geben muss.



Solange ungültige Werte für den Kontext angegeben werden, solange wird der bestehende Kontext nicht verändert. Beim Start des Modules wird ein Standard Kontext gesetzt, was am Anfang der Konsolenausgabe zu sehen ist. Dies bedeutet es kann niemals einen ungültigen oder keinen Kontext geben.

### CreateWorkEntry

Diese Tests testen die Funktion CreateWorkEntry, die eine WorkEntry Instanz erstellt, die als Eintrag in der Liste der Positionen gespeichert wird. Hierbei wird geprüft ob die gegebene TimeSpan Instanz gültig ist, was durch den gesetzten Kontext bestimmt wird.

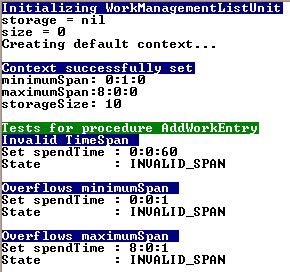


Am Anfang der Konsolenausgabe ist der gesetzte Kontext ersichtlich. Es werden die ungültigen TimeSpan Instanzen korrekt erkannt und auch behandelt. Wenn die TimeSpna Instanz korrekt ist, wird auch ein WorkEntry mit dieser erstellt, andererseits wird der MininumTimeSpan, welcher über den Kontext definiert wurde, auf der WorkEntry Instanz gesetzt um zu verhindern, dass ein undefinierter Wert vorhanden ist.

### AddWorkEntry

Dieser Test testet die Funktion AddWorkEntry, die einen Eintrag in dem Storage speichert, wobei eine neue Person angelegt wird, wenn der Name in der Storage nicht vorhanden ist. Andererseits wird der Eintrag einer gegebenen Person angefügt.

Ungültiger TimeSpan:

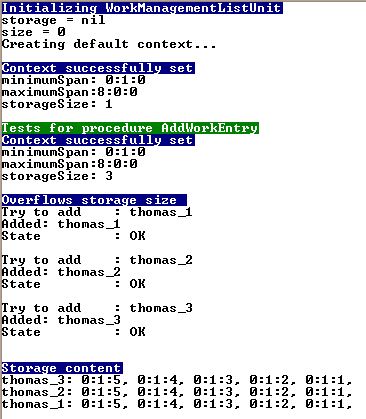


Storage voll:



Es werden alle Einträge für die verschiedenen Personen gespeichert solange noch Platz in dem Storage ist. Ist kein Platz mehr vorhanden wird der Eintrag nicht gespeichert und ein Fehler nach außen bekanntgeben.

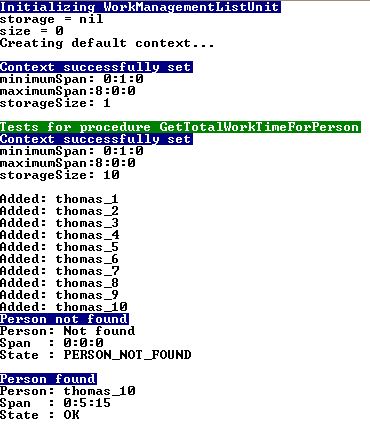
Normales Verhalten:



Sollte es sich um dieselbe Person handeln, so wird der Eintrag in die WorkEntryList der Person hinzugefügt (Anfang der Liste), ansonsten wird eine neue Person angelegt.

### GetTotalWorkTimeForPerson

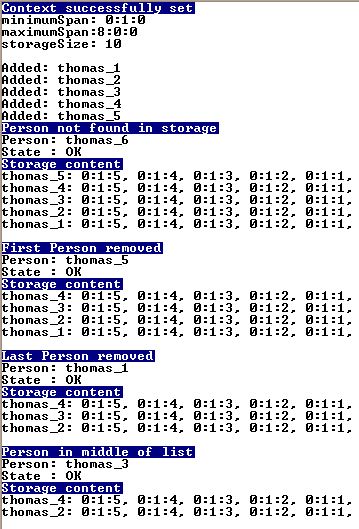
Dieser Test testet die Funktion GetTotalWorkTimeForPerson , die die Gesamtzeit der Arbeitseintrage berechnet und zurückliefert.



Sollte die Person nicht gefunden werden, so wird 0:0:0 zurückgeliefert sowie der dementsprechende Status gesetzt. Ansonsten erfolgt eine korrekte Ausgabe der Summe aller WorkEntry TimeSpan gehaltenen Zeiten einer Person.

### RemovePerson

Dieser Test testet die Funktion RemovePerson, die eine Person sowie alle dazugehörigen Arbeitseinträge löscht.



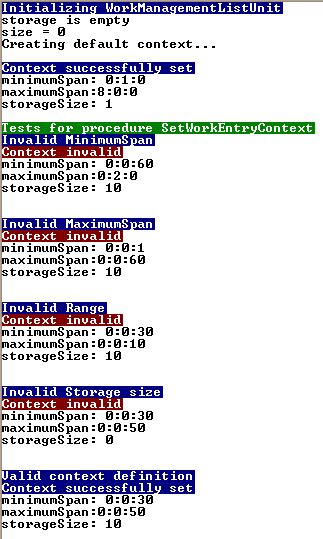
Sollte die Person nicht gefunden werden, so wird der dementsprechende Status zurückgeliefert. Ansonsten wird die Person sowie alle Arbeitseinträge korrekt aus dem Storage entfernt egal ob die Person an erster oder letzter Stelle oder in der Mitte der Liste gefunden wurde.

## HashChained Unit

Folgend sind die Tests der Implementierung WorkManagementHashChainedUnit angeführt, wobei als Storage eine Hashtabelle mit Verkettungsstrategie zur Kollisionsbehandlung verwendet wird. Der Test Source der Single-Linked-List Unit wurde herangezogen und so modifiziert, sodass die Eigenheiten des Storage getestet werden.

### SetWorkEntryContext

Diese Tests testen die Funktion SetWorklfowEntryContext, welche es erlaubt die Unit zur Laufzeit zu konfigurieren, wobei anzumerken ist, das in diesem Fall auch das Storage gelöscht wird. Solange versucht wird einen ungültigen Kontext zu definieren bleibt der zuletzt gültige aktiv. Beim Initialisieren der Unit wird ein Standardkontext gesetzt, da es immer einen aktiven Kontext geben muss..



Solange ungültige Werte für den Kontext angegeben werden, solange wird der bestehende Kontext nicht verändert. Beim Start des Modules wird ein Standard Kontext gesetzt, was am Anfang der Konsolenausgabe zu sehen ist. Dies bedeutet es kann niemals einen ungültigen oder keinen Kontext geben.

### CreateWorkEntry

Diese Tests testen die Funktion CreateWorkEntry, die eine WorkEntry Instanz erstellt, die als Eintrag in der Liste der Positionen gespeichert wird. Hierbei wird geprüft ob die gegebene TimeSpan Instanz gültig ist, was durch den gesetzten Kontext bestimmt wird.

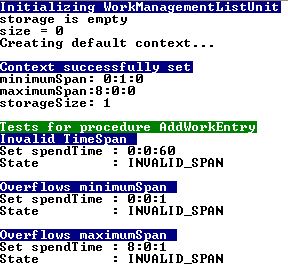


Am Anfang der Konsolenausgabe ist der gesetzte Kontext ersichtlich. Es werden die ungültigen TimeSpan Instanzen korrekt erkannt und auch behandelt. Wenn die TimeSpna Instanz korrekt ist, wird auch ein WorkEntry mit dieser erstellt, andererseits wird der MininumTimeSpan, welcher über den Kontext definiert wurde, auf der WorkEntry Instanz gesetzt um zu verhindern, dass ein undefinierter Wert vorhanden ist.

### AddWorkEntry

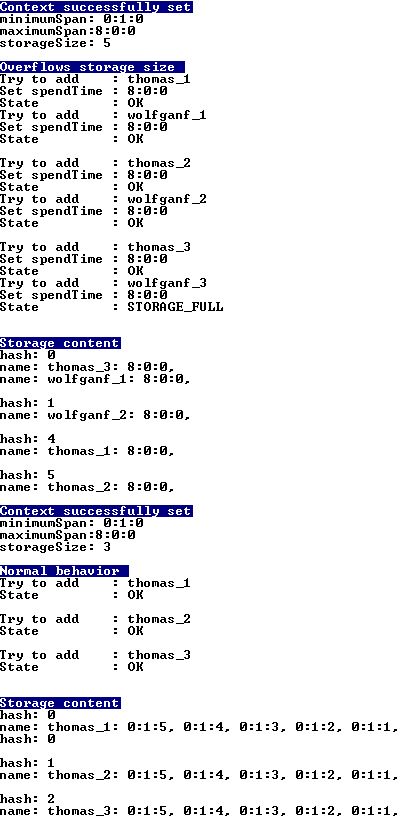
Dieser Test testet die Funktion AddWorkEntry, die einen Eintrag in dem Storage speichert, wobei eine neue Person angelegt wird, wenn der Name in der Storage nicht vorhanden ist. Andererseits wird der Eintrag einer gegebenen Person angefügt. Bei einer Kollision wird die Person der Kette von Personen am berechneten Index hinzugefügt.

Ungültiger TimeSpan:



Solange der Timespan ungültig ist kann der Eintrag nicht gespeichert werden.

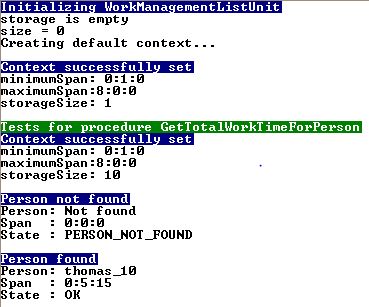
Storage voll:



Hierbei ist auch der berechnete Hashwert ersichtlich der den Index in der Hashtabelle darstellt. Sollte der Storage voll sein, so wird der dementsprechende Status gesetzt. Sollte eine Kollision auftreten, so wird der kollidierende Eintrag der Kette am berechneten Index angefügt.

### GetTotalWorkTimeForPerson

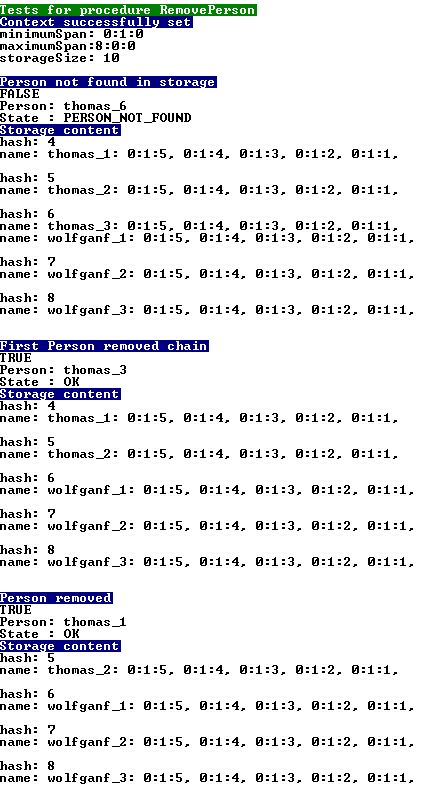
Dieser Test testet die Funktion GetTotalWorkTimeForPerson , die die Gesamtzeit der Arbeitseintrage berechnet und zurückliefert.



Sollte die Person nicht gefunden werden, so wird 0:0:0 zurückgeliefert sowie der dementsprechende Status gesetzt. Ansonsten erfolgt eine korrekte Ausgabe der Summe aller WorkEntry TimeSpan gehaltenen Zeiten einer Person.

### RemovePerson

Dieser Test testet die Funktion RemovePerson, die eine Person sowie alle dazugehörigen Arbeitseinträge löscht.



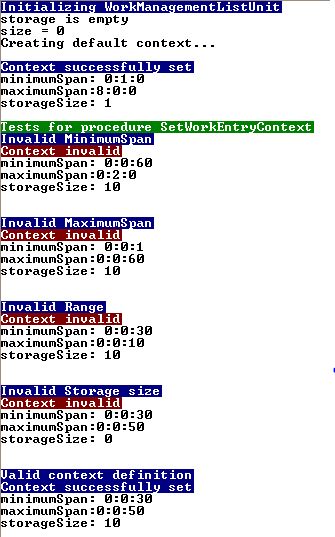
Sollte die Person nicht gefunden werden, so wird der dementsprechende Status zurückgeliefert. Ansonsten wird die Person sowie alle Arbeitseinträge korrekt aus dem Storage entfernt egal ob die Person alleine auf einen Index vorhanden ist oder nicht..

## HashOpen Unit

Folgend sind die Tests der Implementierung WorkManagementHashOpenUnit angeführt, wobei als Storage eine Hashtabelle mit offener Adressierung zur Kollisionsbehandlung verwendet wird. Der Test Source der Single-Linked-List Unit wurde herangezogen und so modifiziert, sodass die Eigenheiten des Storage getestet werden.

### SetWorkEntryContext

Diese Tests testen die Funktion SetWorklfowEntryContext, welche es erlaubt die Unit zur Laufzeit zu konfigurieren, wobei anzumerken ist, das in diesem Fall auch das Storage gelöscht wird. Solange versucht wird einen ungültigen Kontext zu definieren bleibt der zuletzt gültige aktiv. Beim Initialisieren der Unit wird ein Standardkontext gesetzt, da es immer einen aktiven Kontext geben muss..



Solange ungültige Werte für den Kontext angegeben werden, solange wird der bestehende Kontext nicht verändert. Beim Start des Modules wird ein Standard Kontext gesetzt, was am Anfang der Konsolenausgabe zu sehen ist. Dies bedeutet es kann niemals einen ungültigen oder keinen Kontext geben.

### CreateWorkEntry

Diese Tests testen die Funktion CreateWorkEntry, die eine WorkEntry Instanz erstellt, die als Eintrag in der Liste der Positionen gespeichert wird. Hierbei wird geprüft ob die gegebene TimeSpan Instanz gültig ist, was durch den gesetzten Kontext bestimmt wird.

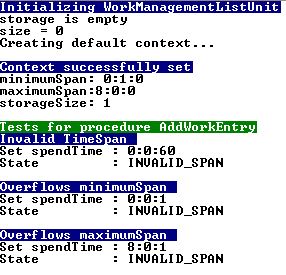


Am Anfang der Konsolenausgabe ist der gesetzte Kontext ersichtlich. Es werden die ungültigen TimeSpan Instanzen korrekt erkannt und auch behandelt. Wenn die TimeSpna Instanz korrekt ist, wird auch ein WorkEntry mit dieser erstellt, andererseits wird der MininumTimeSpan, welcher über den Kontext definiert wurde, auf der WorkEntry Instanz gesetzt um zu verhindern, dass ein undefinierter Wert vorhanden ist.

### AddWorkEntry

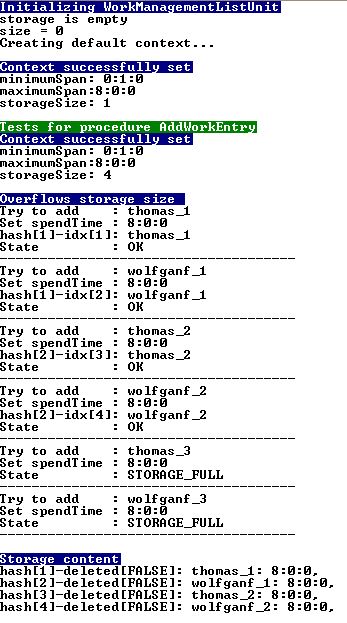
Dieser Test testet die Funktion AddWorkEntry, die einen Eintrag in dem Storage speichert, wobei eine neue Person angelegt wird, wenn der Name in der Storage nicht vorhanden ist. Andererseits wird der Eintrag einer gegebenen Person angefügt. Bei einer Kollision wird ein freier Index gesucht wo die Person abgelegt werden kann. Die Hashtabelle wird dabei vollständig gesucht, was bedeutet, dass wenn das Ende der Tabelle erreicht wurde am Anfang der Tabelle die Suche fortgesetzt wird, solange bis der berechnete Index erreicht wurde. In diesem Fall könnte die Person nicht gespeichert werden.

Invalid Timespan:



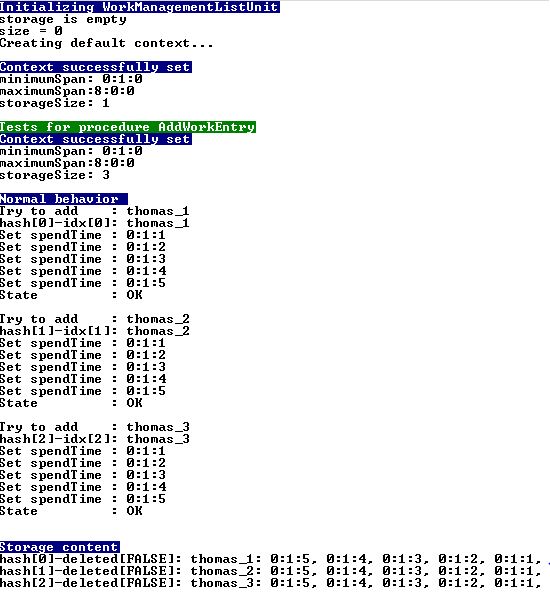
Solange der Timespan ungültig ist kann der Eintrag nicht gespeichert werden.

Storage voll:



Es werden alle Einträge für die verschiedenen Personen gespeichert solange noch Platz in der Storage ist. Ist kein Platz mehr vorhanden wird der Eintrag nicht gespeichert und ein Fehler nach außen bekanntgeben. Hierbei ist zu sehen, dass die gespeicherten Personen ein deleted Flag besitzen, da sie aus dem Storage nicht vollständig gelöscht werden dürfen, dies ist begründet durch die Art der Kollisionsbehandlung.

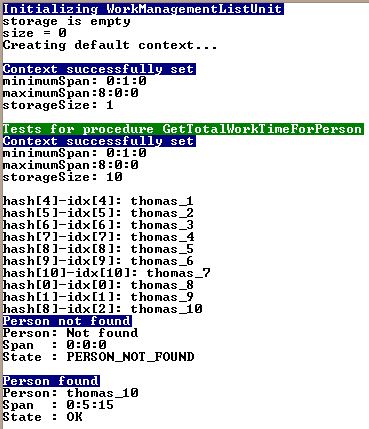
Keine Fehler:



Solange das Storage nicht gefüllt ist, wird die Person gespeichert.

### GetToalWorkTimeForPerson

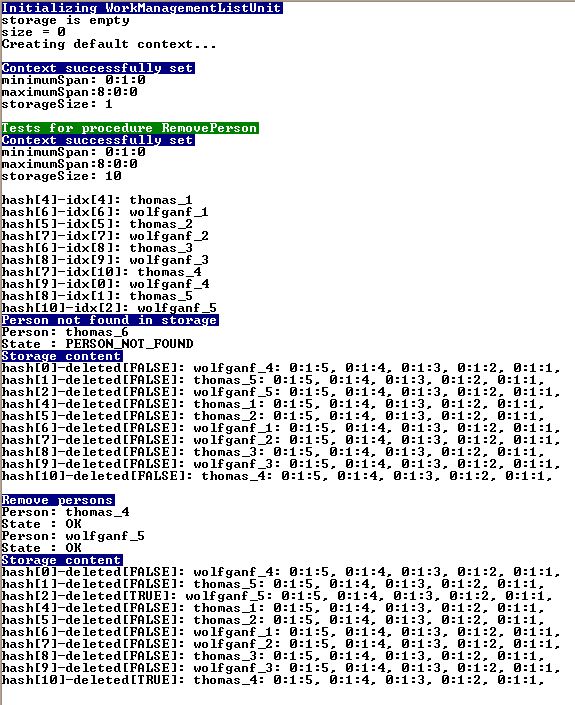
Dieser Test testet die Funktion GetTotalWorkTimeForPerson , die die Gesamtzeit der Arbeitseintrage berechnet und zurückliefert.



Sollte die Person nicht gefunden werden, so wird 0:0:0 zurückgeliefert sowie der dementsprechende Status gesetzt. Ansonsten erfolgt eine korrekte Ausgabe der Summe aller WorkEntry TimeSpan gehaltenen Zeiten einer Person. Hierbei ist auch ein Beispiel angeführt, wo eine Person Aufgrund einer Kollision an anderen einen freien Index gespeichert werden musste und das sollte das Ende der Tabelle erreicht worden sein von Anfang bis zum berechneten Index ein freier Index gesucht wird. Hierbei wird der Index mit (newHash = hash + 1) gesucht. Es wäre auch möglich gewesen diesen mittel quadratischer Funktion (newhash = hash^2 mod Indexrange) zu suchen was vielleicht eine höhere Wahrscheinlichkeit einen freien Index zu finden hätte.

### RemovePerson

Dieser Test testet die Funktion RemovePerson, die eine Person sowie alle dazugehörigen Arbeitseinträge löscht.



Sollte die Person nicht gefunden werden, so wird der dementsprechende Status zurückgeliefert. Ansonsten wird die Person sowie alle Arbeitseinträge korrekt aus dem Storage entfernt egal ob die Person an einen anderen Index gespeichert wurde, als der der berechnet wurde. Es wird hierbei lediglich das deleted Flag gesetzt, da die Person nicht vollständig gelöscht werden darf.

# Diskussion

Wenn als Storage eine einfach verkette Liste verwendet wird, muss man sich keine Sorgen um die Anzahl der gespeicherten Einträge machen, da diese Datenstruktur dynamisch ist und aus Performancegründen die Personen immer am Anfang der Liste angefügt werden, was beim Anfügen ein konstantes Laufzeitverhalten aufweist. Das Problem besteht darin, dass bei einer Suche im worst case Fall alle Elemente der Liste besucht werden müssen, ebenso beim Löschen einer Person. Im Gegensatz zu einer Hashtabelle wo der Aufwand beim berechnen des Hashwertes liegt und der Zugriff immer ein konstantes Laufzeitverhalten aufweist, was durch den indexierten Zugriff möglich ist.

Bei Verwendung einer Hashtabelle mit Verkettung als Kollisionsbehandlung besteht das Problem das die Elemente, die denselben Hashwert haben wieder in einer einfach verketteten Liste gespeichert werden und daher bei diesen Elementen die Probleme , die eine einfach verkette Liste mit sich bringt, auftreten. Jedoch muss hier nicht nach einen neuem Index gesucht werden, was bei einer großen Hashtabelle, die einen hohen Füllfaktor hat dazu führen kann, das nahezu die gesamte Tabelle nach einen Index durchsucht werden muss.

Bei offener Adressierung muss man nach einen neuen Index gesucht werden wo das Element abgelegt werden kann und die Elemente die einmal in die Tabelle eingetragen wurden, dürfen solange die Tabelle verwendet wird, nicht vollständig gelöscht werden. Man muss hier einen eigenen Datentyp definieren oder den Element Datentypen ein Attribut vergeben welches Sie als gelöscht markiert. Es muss hierbei beachtet werden das nil bedeutet, es gibt einen freien Index und andererseits ist das Element nur als gelöscht markiert. In den beiden Fällen der Kollisionsbehandlung ist immer mit einen Mehraufwand zu rechnen, daher wäre es besser Kollisionen ganz zu vermeiden, was dadurch zu realisieren wäre, indem man einen Hashalgorithmus verwendet, welcher immer gleichverteilte Indizes liefert. In diesem Beispiel wäre es ein Hashalgorithmus, welcher nicht nur den Ordinalwert des Zeichens sondern ebenso seine Position im String miteinbezieht und ebenso sollte die Hashtabelle möglichst groß gewählt werden, um einen großen Adressraum zur Verfügung zu haben (Maximum Füllfaktor 0,8). Dieser Algorithmus wurde in dieser Übung von mir nicht gewählt um bewusst Kollisionen zu verursachen um diese auch korrekt behandeln zu können.