Computational Logic 2015
First Optional Homework
Student: ***Vlad Cîncean***
Group: **913**

0 Contents

1 Problem Statement

Write an application that implement algorithms for:

- Arithmetic operations: *addition*, *subtraction*, *multiplication* and *division* by one digit, in a base $p \in \{2,3, \dots ,10,16\}$
- Conversions of natural numbers between two bases $p, q \in \{2,3, \dots ,10,16\}$ using the *substitution method* or *successive divisions* and *rapid conversions* between two bases $p, q \in \{2,4,8,16\}$

The application must have a menu such that all operations and conversion methods to be verified separately.

The executable form, the code of the application and the documentation will be written on a CD (one CD for each group, in which every student will have a personal folder). The documentation will follow the same structure as the Programming Fundament's documentations, and it must contain at least: the problem statement for the implemented application, the used algorithms in pseudo-code, implementation considerations and test data.

The mark is computed as follows:

10%: by default

70%: the application (the authors name will be found in code and will be visible at run too)

- 1p - algorithm for the method of successive divisions
- 1p - algorithm for the substitution method
- 1p – algorithm for conversion using 10 as an intermediate base
- 2p - rapid conversions (executable form) between two bases $p, q \in \{2,4,8,16\}$
- 1p addition of two numbers in a base
- 1p subtraction of two numbers in a base
- 1p multiplication of a number by a digit in a base
- 1p division of a number by a digit in a base
- 1p code quality (indentation, use of comments, suggestive variables names)

20%: documentation

- 1p problem statement
- 1p sub-algorithm's diagram
- 1p used data type specification
- 3p specification and pseudo-code for the important algorithms used (input, output, preconditions, post-conditions -1p; pseudo-code 2p)
- 3p at least a set of test data for the complete application, more data sets where is needed
- 1p documentation clearness (structured, well written …)

2 Feature List

Category	Feature	Description
Operations	add	Performs an *addition* between two numbers in base $p \in \{2,3,\dots,10,16\}$
	sub	Performs a *subtraction* between two numbers in base $p \in \{2,3,\dots,10,16\}$
	mul	Performs the *multiplication* of a number by a digit in base $p \in \{2,3,\dots,10,16\}$
	div	Performs the *division* of a number by a digit in base $p \in \{2,3,\dots,10,16\}$
Conversions	substitution method	Performs the *conversion* of a number from the base $s \in \{2,3,\dots,10,16\}$ to the destination base $d \in \{2,3,\dots,10,16\}$ $(s < d)$ using the *substitution method*
	successive divisions method	Performs the *conversion* of a number from the base $s \in \{2,3,\dots,10,16\}$ to the destination base $d \in \{2,3,\dots,10,16\}$ $(d < s)$ using the *successive divisions method*
	rapid conversion	Preforms the *rapid conversion* of a number between two bases – powers of 2 $s, d \in \{2,4,8,16\}$

3 Running Scenarios

3.1 Operations

#	User	Program	Description
01		<welcome message>	The program prints on the screen a welcome message and gives the user instructions about how to enter the input data.
02		*The base p =*	The program asks the user to enter the base in which he want the computations to be made.
03	*16*		The user chooses base 16, for example.
04		*Expression =*	The program asks the user to input the expression of a binary operation in base 16.
05	*4A20+C9*		The user inputs the expression of an operation.
06		*4A20 + C9 = 4AE9*	The program gives the result.
07		<?>	The program asks the user if (s)he want to perform another operation in base 16.
08	*yes*		The user answers affirmative.
09		<go to #04>	Similar scenario to steps 04-07.
10	*no*		The user answers negative.
11		<main menu>	The program returns to the main menu.

3.2 Conversions

#	User	Program	Description
01		<welcome message>	The program prints on the screen a welcome message and gives the user instructions about how to enter the input data.
02		*The source base =*	The program asks the user to enter the *source* base from which he want to convert.
03	*10*		The user inputs the source base *10*, for example.
04		*The destination base =*	The program asks the user to enter the *destination* base in which he want to convert.
05	*6*		The user inputs the destination base *6*, for example.
06		*n =*	The program asks the user to enter the number's *representation* in the source base.
07	*862*		The user inputs *862*, for example.
08		$826_{10} = 3554_6$	The program prints the result.
09		<the method used>	The program prints the method used. It is the *substitution method* if $s < d$ and the *successive divisions method* if $d < s$.
10		<main menu>	The program return to the main menu.

3.3 Rapid Conversions

#	User	Program	Description
01		<welcome message>	The program prints on the screen a welcome message and gives the user instructions about how to enter the input data.
02		*The source base =*	The program asks the user to enter the *source* base from which he want to convert.
03	*16*		The user inputs the source base *16*, for example.
04		*The number to convert from base 16 =*	The program asks the user to enter the number's *representation* in base 16 to convert.
05	*A2D7*		The user inputs *A2D7*, for example.
06		*The destination base =*	The program asks the user to enter the *destination* base in which he want to convert.
07	*2*		The user inputs the destination base *2*, for example.
08		$A2D7_{16} =$ $= 1010001011010111_2$	The program prints the result
09		*Another one?*	The program asks the user if (s)he want to perform another *rapid conversion*.
10	*yes*		The user answers affirmative.
11		<go to #02>	Similar scenario to steps 02-09.
12	*No*		The user answers negative.
13		<main menu>	The program returns to the main menu.

4 Operations – Work Items/Tasks

#Task	Operations
01	**add** – adds two numbers in base *p*
02	**sub** – subtracts two numbers in base *p*
03	**mul** – multiplies a number *A* in base *p* by a digit *b* in base *p*
04	**div** – divides a number *A* in base *p* by a digit *b* in base *p*

4.1 Addition

4.1.1 Specification

```
Adds 2 numbers in base p
Input: a (string) - the augend's representation in base p
       b (string) - the addend's representation in base p
       p (integer) - the base (2, 3, ..., 10 or 16)
Output: c (string) - the representation in base p of the result of the multiplication
            c = a + b
```

4.1.2 Testing

#	Input			Output
	a	**b**	**p**	
01	1010	110	2	10000
02	222	7	10	229
03	7	222	10	229
04	23	14	5	42
05	2AF6	12C	16	2C22
06	45	0	7	45
07	0	45	7	45

4.1.3 Implementation

```
def add(a, b, p):
    '''1. we reverse the order of the digits in both numbers' representations '''
    a = reverse(a)
    b = reverse(b)
    '''2. we add insignificant zeros where it is needed '''
    if len(a) > len(b):
        a += '0'
        while len(a) > len(b):
            b += '0'
    elif len(a) < len(b):
        b += '0'
        while len(a) < len(b):
            a += '0'
    elif len(a) == len(b):
        a += '0'
        b += '0'
    '''3. we initialize a 'transport string', which will store the transport digits for all
addition iterations '''
    t = '0' #the first transport digit is '0' (zero)
```

```
    '''4. the addition process '''
    c = ''
    for i in range(0, len(a), 1):
        s = to_value(a[i]) + to_value(b[i]) + int(t[i])
        t += str(s // p) #it can be '0' or '1'
        c += to_repr(s % p)
    '''5. we remove the insignificant zero from the front of the result's representation in
base p, if necessary '''
    while c[-1] == '0' and len(c) > 1:
        c = c[:-1]
    '''6. we reverse back the order of the digits in the result's representation '''
    c = reverse(c)
    '''7. we return the result '''
    return c
```

4.2 Subtraction

4.2.1 Specification

```
    Subtracts 2 numbers in base p
    Input: a (string) - the minuend's representation in base p
           b (string) - the subtrahend's representation in base p
               condition: a >= b
               - returns ValueError in case this condition is not meet
           p (integer) - the base (2, 3, ..., 10 or 16)
    Output: c (string) - the representation in base p of the result of subtraction
                c = a - b
```

4.2.2 Testing

#	Input			Output
	a	b	p	
01	10101011	11001	2	10010010
02	84	9	10	75
03	3C8	0	16	3C8
04	24	24	7	0

4.2.3 Implementation

```
def sub(a, b, p):
    '''1. we reverse the order of the digits in both numbers' representations '''
    a = reverse(a)
    b = reverse(b)
    '''2. we add insignificant zeros where it is needed '''
    while len(b) < len(a):
        b += '0'
    '''3. we initialize a 'transport string', which will store the transport digits for all
subtraction iterations '''
    t = '0' #the first transport digit is '0' (zero)
    '''4. the subtraction process '''
    c = ''
    for i in range(0, len(a), 1):
        if to_value(a[i]) - int(t[i]) < to_value(b[i]):
            t += '1'
        else:
            t += '0'
        s = p*int(t[i+1]) + to_value(a[i]) - int(t[i]) - to_value(b[i])
        c += to_repr(s)
```

```
    '''5. we remove the insignificant zeros from the front of the result's representation in
base p, if necessary '''
    while c[-1] == '0' and len(c) > 1:
        c = c[:-1]
    '''6. we reverse back the order of the digits in the result's representation '''
    c = reverse(c)
    '''7. we return the result '''
    return c
```

4.3 Multiplication

4.3.1 Specification

```
    Multiplies a number A in base p by a digit b in base p
    Input: A (string) - the number's representation in base p
           b (character) - the digit's representation in base p
           p (integer) - the base (2, 3, ..., 10 or 16)
    Output: c (string) - the representation in base p of the result of the multiplication
                c = A * b
```

4.3.2 Testing

#	Input			Output
	A	b	p	
01	19	B	16	113
02	ABC	D	16	8B8C
03	256	3	10	768
04	10111	1	2	10111
05	10111	0	2	0

4.3.3 Implementation

```
def mul(A, b, p):
    '''1. we reverse the order of the digits in the number's representations '''
    A = reverse(A)
    '''2. we add a insignificant zero in the front of A's representation in base p '''
    A += '0'
    '''3. we initialize a 'transport string', which will store the transport digits for all
multiplication iterations '''
    t = '0' #the first transport digit is '0' (zero)
    '''4. the multiplication process '''
    c = ''
    for i in range(0, len(A), 1):
        s = to_value(A[i]) * to_value(b) + to_value(t[i])
        t += to_repr(s // p)
        c += to_repr(s % p)
    '''5. we remove the insignificant zero from the front of the result's representation in
base p, if necessary '''
    while c[-1] == '0' and len(c) > 1:
        c = c[:-1]
    '''6. we reverse back the order of the digits in the result's representation '''
    c = reverse(c)
    '''7. we return the result '''
    return c
```

4.4 Division

4.4.1 Specification

```
Divides a number A in base p by a digit b in base p
Input: A (string) - the dividend's representation in base p
       b (character) - the digit's representation in base p
           b <> 0
       p (integer) - the base (2, 3, ..., 10 or 16)
Output: q (string) - the representation in base p of the quotient of the division
        r (character) - the representation in base p of the remainder of the division
            q remainder r = A / b
```

4.4.2 Testing

#	Input			Output
	a	b	p	
01	9420	7	10	1345 r 5
02	2AB4F	B	16	3E1E r 5
03	10110	1	2	10110 r 0
04	858	0	9	ValueError

4.4.3 Implementation

```
def div(A, b, p):
    if b == '0':
        raise ValueError("Cannot divide by zero!")
    '''1. we initialize a 'transport string', which will store the transport digits for all
division iterations '''
    t = '0' #the first transport digit is '0' (zero)
    '''2. the division process '''
    q = ''
    for i in range(0, len(A), 1):
        q += to_repr((int(t[i]) * p + to_value(A[i])) // to_value(b))
        t += str((int(t[i])*p+to_value(A[i]))-
(((int(t[i])*p+to_value(A[i])))//to_value(b))*to_value(b))
    r = t[-1]
    '''3. we remove the insignificant zero from the front of the quotient's representation in
base p, if necessary'''
    while len(q) > 1 and q[0] == '0':
        q = q[-len(q)+1:]
    '''4. we return the result '''
    return q, r
```

5 Conversions – Work Items/Tasks

5.1 Substitution Method

5.1.1 Specification

```
    Performs a conversion from base s to base d using the substitution method
    It is recommended if s < d ! Raises ValueError if s >= d or s,d not in {2, 3, ...,
10, 16}
    Input: n (string) - the number's representation in base s
           s (integer) - the source base
           d (integer) - the destination base)
    Output: res (string) - the number's representation in base d
```

5.1.2 Implementation

```
def substitution_method(n, s, d):
    if s >= d or not(s >= 2 and s <= 10 or s == 16) or not(d >= 2 and d <= 10 or d == 16):
        raise ValueError("Cannot convert using substitution method!")
    ''' if s < d, then all the digits of the number's representation in base s are the same
with those read in base d '''
    ''' if s < d, then s(d) = s (the source base represented in the destination base) '''
    ''' we compute the result '''
    res = '0'
    for i in range(len(n)-1, -1, -1):
        t = '1'
        for j in range(0, len(n)-1-i, 1):
            if s == 10:
                t = mul(t, 'A', d)
            else:
                t = mul(t, str(s), d)
        t = mul(t, n[i], d)
        res = add(res, t, d)
    ''' we return the result '''
    return res
```

5.1.1 Testing

#	Input			Output
	a	s	d	
01	345	6	8	216
02	11011	2	4	123

5.2 Successive Divisions Method

5.2.1 Specification

```
    Performs a conversion from base s to base d using the successive divisions method
    It is recommended if s > d ! Raises ValueError if s <= d or s,d not in {2, 3, ...,
10, 16}
    Input: n (string) - the number's representation in base s
           s (integer) - the source base
           d (integer) - the destination base)
    Output: out (string) - the number's representation in base d.
```

5.2.2 Implementation

```
def successive_divisions_method(n, s, d):
    if s <= d or not(s >= 2 and s <= 10 or s == 16) or not(d >= 2 and d <= 10 or d == 16):
        raise ValueError("Cannot convert using successive divisions method!")
    q = n
    res = ''
    while q != '0':
        if d == 10:
            q, r = div(q, 'A', s)
        else:
            q, r = div(q, str(d), s)
        res = r + res
    ''' we delete the insignificant zeros from the front of the result, if necessary '''
    while len(res) > 0 and res[0] == '0':
        res = res[:-1]
    ''' we return the result '''
    return res
```

5.2.3 Testing

#	Input			Output
	a	s	d	
01	24	10	2	11000
02	A5B	16	8	5133

5.3 Rapid Conversions

5.3.1 Specification

```
    Performs a rapid conversion between two bases as powers of 2 (2, 4, 8 or 16)
    Input: n (string) - the number's representation in base s
           s (integer) - the source base
           d (integer) - the destination base
    Output: out (string) - the representation in base d of the number
```

5.3.2 Implementation

```
def to_base_2(n, s):
    if s != 2 and s != 4 and s != 8 and s != 16:
        raise ValueError("Error! Cannot perform a rapid conversion from the source base " + s)
    MAP = {'0':'0000', '1':'0001', '2':'0010', '3':'0011', '4':'0100', '5':'0101', '6':'0110',
'7':'0111','8':'1000', '9':'1001', 'A':'1010', 'B':'1011', 'C':'1100', 'D':'1101', 'E':'1110',
'F':'1111'}
    ''' we compute the result '''
    out = ''
    for digit in n:
        out += MAP[digit][-int(log2(s)):]
    ''' we remove the insignificant zeros from the front of the result '''
    while len(out) > 1 and out[0] == '0':
        out = out[1:]
    return out
```

```
def from_base_2(n, d):
    if d != 2 and d != 4 and d != 8 and d != 16:
        raise ValueError("Error! Cannot perform a rapid conversion to the destination base " +
d)
    MAP = {'0000':'0', '0001':'1', '0010':'2', '0011':'3', '0100':'4', '0101':'5', '0110':'6',
'0111':'7','1000':'8', '1001':'9', '1010':'A', '1011':'B', '1100':'C', '1101':'D', '1110':'E',
'1111':'F'}
    ''' we add insignificant zeros in the front of the number's representation in base 2 '''
    while len(n)%int(log2(d)) != 0:
        n = '0' + n
    ''' we compute the result '''
    out = ''
    for i in range(0, len(n), int(log2(d))):
        out += MAP[(4-int(log2(d)))*'0' + n[i:i+int(log2(d))]]
    ''' we remove the insignificant zeros from the front of the result '''
    while len(out) > 1 and out[0] == '0':
        out = out[1:]
    return out

def rapid_convert(n, s, d):
    out = from_base_2(to_base_2(n, s), d)
    return out
```

5.3.3 Testing

#	Input			Output
	a	s	d	
01	1101100	2	4	1230
02	10100111000001	2	8	24701
03	1230	4	8	154
04	33220213	4	16	FA27

6 Miscellaneous

```
def to_value(d):
    '''
    Converts a digit represented in any base (2, 3, ..., 10 or 16) to its decimal value
    Input: d (character) - the digit in any base
    Output: v (integer) - d's decimal value
    '''
   .........
def to_repr(v):
    '''
    Converts a decimal value (0, 1, ..., 15) to its representation in a numerical base (max 16)
    Input: v (integer) - the decimal value
    Output: d (character) - the representation in base 16 (or lower) of the given decimal value
    '''
   .........
def reverse(st):
    '''
    Returns the given string 'st' in the reverse order
    e.g. if st = 'abc12', then '21cba' is returned
    Input: st (string)
    Output: st2 (string)
    '''
   .........
```