**Implementation operations in different bases**

**and**

**Conversion of numbers from a base to another**

Author: Tasadan Filip

1. **Problem statement**

Implement algorithms for:

1. v arithmetic operations for positive integers: addition, subtraction, multiplication and division by one digit, in a base p €{2,3,...,9,10,16}
2. v conversions of natural numbers between two bases p, q € {2,3,...,9,10,16} using the substitution method or successive divisions and rapid conversions between two bases p, q €{2, 4, 8, 16}.

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

1. **Implementation considerations**

The project is written in 'Python 3' programming language and it's splitted in four separate modules, every module having a specific role in the way the program works. Every module contains a class where are stored the used variables and the methods (functions) everyone having a particular role in that class. The methods inside the classes were written with respect to the "Single responsability" principle (every method has responsibility over a single part of the program's functionality).

Note: the menu will be displayed after every operation in order to simplify the way of accessing a particular functionality (the result of the requested operation or conversion is going to be on the next line after the input data).

1. **Modules specification and pseudo-code**

The 'main.py' function was not classified as a module because it contains only the initialization and the run command of the User Interface.

In the following paragraphs I am going to explain each separate module ( what it does and take the most important methods inside it and explain it using both natural language and pseudo-code).

1. **Console module**

This module contains the User Interface class where all the input is introduced into the program and the output data is displayed to the console. For each separate functionality we have a corresponding digit which should be specified in order to access that particular functionality ( Example: if you want to add two numbers you should type '1' in order to access the 'add functionality' inside the program). In this class we have a special method called 'run' which represent the main loop of this program. This loop will end only when the user type the 'exit' command or the console is closed. If the user enter an invalid command or number the program will display an 'Error' message but the program loop is not going to crash.

Used data type specification can be seen in this class by looking at the first two methods from it:

1. 'operations\_input' contains the specific functions which will transfer the data typed in the console by the user into the programf for performing the four main operations (addition, subtraction, multiplication and division). The user is going to introduce three values (for each separate operation):

- The first value is the base (in decimal) in which he wants the numbers to be computed

- The second value is the first operand (in decimal) of that particular operation

- The third value is the second operand (in decimal) of that particular operation

These three numbers are going to be interpreted as a string of digits in the given base but they are going to be transformed as list of decimal values, each one representing a digit of this string ( more information about this will be in the presentation of number\_entity module').

1. 'conversions\_input' contains the specific functions which will transfer the data typed in the console by the user into the program for the four types of conversion methods (substitution method, successive division method, intermediate base 10 conversion method and rapid conversion method) the user is going to introduce three values:

- The first value is the number (in the source base) he wants to be converted by the program

- The second value is the source base (in decimal)

- The third value is the destination base (in decimal)

These three numbers are going to be interpreted as a string of digits in the given base but they are going to be transformed as list of decimal values, each one representing a digit of this string ( more information about this will be in the presentation of 'Conversions module').

1. **number\_entity module**

This module contains the 'Number' class which will serve 2 different purposes. The first one is to check the numbers which were introduced are valid in the given base (Example: 125 is a valid number in base 6 but it is not valid in base 2) and to issue a specific error if they are not valid. The second purpose of is to transform the numbers from string of digits into list of decimal numbers, each one representing a specific digit of that string (Example: the string '12BC' which represent the number 12BC in base 16 will be transformed into the following list: [1, 2, 10, 11]).

1. **Operations module**

This module contains the 'NumberOperations' class which encapsulate the four methods (functions, do not be confused with conversion method) that contain the algorithm of operations with numbers in a given base. Here is the actual place where addition, subtraction, multiplication and division of the numbers take place.

**Important note:** 3.1)For every operation the two operands and the base are represented as list of decimal digits in order to be easier to work with them.

3.2) All the operations algorithms follow the same structure as those presented at the first two seminars in the start of the semester. In other words, what I did in this project was to transform the pseudo-code algorithms from seminar into the 'Python 3' working code.

1. **Addition of two numbers**

Pseudo-code:

c0=0;        //c0, c1,… ,cn+1€ {0,1}are the carries used in addition

         for i=0,n

              x’i (10)= xi (b) ;    y’i (10)= yi (b);    //convert the digits from base b in base 10

s10 = x’i (10)+y’i (10) +ci (10);      //addition in base 10

              s’i (10)= s10 mod b ;   ci+1= s10 div b;

              si (b) = s’i (10);                       //convert the decimal value in base b

          end\_for

              si+1 = ci+1;

Explanation of the algorithm:

The first step is to check which is the longest number in order to add the 0 digit in front of the smaller one in order to have the same length for the two lists which represent the two operands. In order to have the same length for the list, I inverted both lists and appended at the end of the smaller one 0's until the two lists will have the same length. After this, I parse the two lists and compute the value of s10 (from the pseudo-code) as follows: the carry digit from that position added with the digit (transformed in decimal) of the first operand and with the digit (transformed in decimal) of the second operand. The result (s10) is divided by 10 and the remainder value is going to be putted to the result list and the quotient will represent the carry for the digit on the next position. In the end we will obtain a result list which will contain the digits (in decimal) of the result of the addition and every number from this list is going to be converted in the given base ( Ex: 10 is going to be converted in A in base 16) and transformed into a string of digits in that given base.

1. **Subtraction of two numbers**

Pseudo-code:

c0=0; //c0, c1,… ,cn+1€ {0,1}are the borrows used in subtraction

for i=0,n

              x’i (10)= xi (b) ;    y’i (10)= yi (b);    //convert the digits from base b in base 10

              d10 = x’i(10)– ci (10)–y’i (10);     //subtraction in base 10

    ci+1= 0;

    if (d10 < 0 )  then   { d10 = d10+b ;  ci+1= 1;}

    di (b)= d10(10)                             //convert the decimal value in base b

  end\_for

Explanation of the algorithm:

The first step is to check which is the longest number in order to add the 0 digit in front of the smaller one in order to have the same length for the two lists which represent the two operands. In order to have the same length for the list, I inverted both lists and appended at the end of the smaller one 0's until the two lists will have the same length. After this, I parse the two lists and compute the value of d10 (from the pseudo-code) as follows: the value of the first digit in decimal of the first operand from that position is subtracted with the value of the borrow from that position and the value of the second operand digit in decimal. If d10 is smaller than 0 the borrow value from the next position will be incremented with 1 and d10 will take the value of -d10 (or it's modulus value). The value of the result list on the position where the last computations were made will be the modulus of d10. In the end we will obtain a result list which will contain the digits (in decimal) of the result of the subtraction and every number from this list is going to be converted in the given base ( Ex: 10 is going to be converted in 'A' in base 16) and transformed into a string of digits in that given base.

1. **Multiplication of two numbers (one of them having one digit)**

Pseudo-code:

  c0=0;        //c0, c1,… ,cn+1are the carries used in multiplication

         for i=0,n

              x’i (10)= xi (b) ;    f ’(10)= f(b);    //convert the digits from base b in base 10

p10 = x’i (10)\*f ’(10) +ci (10);    //operations in base 10

              p’i (10)= p10 mod b ;   ci+1= p10 div b;

              pi (b) = p’i (10);                    //convert the decimal value in base b

          end\_for

              pi+1 = ci+1;

Explanation of the algorithm:

The first step was to inverse the list of the operand with a length bigger than 1 in order to parse the digits from the least significant one to the most significant. After this the digit of the value of p10 (in decimal) from a particular position is: the digit of the longer operand from this position multiplied with the digit of the 1-digit operand and at this the carry value from this position is added. The value from the particular position (in which the program computed p10) is going to be the remainder of the division between the value of p10 and the value of the given base (both in decimal) and the carry value from the next position will be the quotient of this division. In the end we will obtain a result list which will contain the digits (in decimal) of the result of the subtraction and every number from this list is going to be converted in the given base ( Ex: 10 is going to be converted in 'A' in base 16) and transformed into a string of digits in that given base.

1. **Division of two numbers (the second one having one digit)**

Pseudo-code:

r’ = 0; // r’ – a decimal value

for i = n,0,-1

               x’i (10)= xi (b) ;              //convert the digits from base b in base 10

               p10 = r’\*b +  x’i (10);  //operations in base 10

               q’i (10)= p10 div f ;      r’= mod f;

     qi (b)= q’i (10)//convert the decimal value in base b

  end\_for

r(b) =r’(10)//convert the decimal value in base b

Note: because the used algorithm for the division has the exact same structure as this pseudo-code algorithm I don't think it requests additional explanations.

1. **Conversions module**

This module contains the ' NumberConversions' class which encapsulate the four methods (functions, do not be confused with conversion method) that contain the algorithm of conversions of a number from a source base to a destination one. Here is the actual place where the conversions are made in this program.

**Important note:** 4.1)For every conversion the value of the number to be converted is represented as list of decimal digits in order to be easier to work with it.

4.2) All the conversion algorithms follow the same structure as those presented at the first two seminars in the start of the semester. In other words, what I did in this project was to transform the algorithms from seminar into the 'Python 3' working code.

4.3) All the conversion algorithms use the previously implemented operations to perform the required operations.

1. **Substitution method**

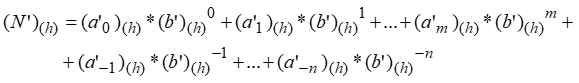
* calculations are made in the destination base
* it is recommended for *b < h*, *b* (source base),  *h*(destination base)

**Important note: the algorithm will work even if the b >= h**

*Steps:*

* all the digits from the source representation are converted into the destination base: 
* the base*b* is converted into base *h*:  
* we calculate in base *h* the following sum:

****

Code explanation:

1. 'positional\_power': represent the actual value of (b')(h) ^ i which will represent one of the operands for the multiplication with the value of the digit (of the number to be converted) on that paricular position. This power will multiply with the value of source base (in the destination base) for every digit in order to obtain the right positional power of that value
2. 'positional\_number': represent the value of the multiplication of the positional digit (of the number to be converted) with the positional power, this operation is made in the destination base
3. 'converted\_number': this variable is nothing else than the sum of all

positional digits value multiplied with the positional power, this operation is made in the destination base and every time we calculate a new factor of this sum we add it to the old value of the 'converted\_number'

Result: in the end we will convert the 'converted\_number' (it is a list of digits) into a string of digits which will be returned to the 'Console module' to be displayed on the console

1. **Successive division method**

* it is recommended for *h < b*, *b* –source base and *h*- destination base.

**Important note: the algorithm will work even if the h >= b**

How the algorithm works:

- calculations are made in the source base

- **successive divisions** by the destination base (***h***) are performed

* the process of successive divisions ends when 0 is obtained as quotient.
* the remainders, in the reverse order, are the digits of the new representation in base *h.*

1. **Conversion using base 10 as intermediate method**

This function will use the last 2 implemented conversions in order to execute the required conversion. The algorithm will respect the following rules:

- If the source base is smaller than 10 the program will convert from source base to base 10 using successive division method

- If the source base is bigger than 10 the program will convert from source base to base 10 using substitution method

- If the destination base is smaller than 10 the program will convert from base 10 to destination base using successive division method

- If the destination base is bigger than 10 the program will convert from base 10 to destination base using substitution method

Result: it will return the value of our number in the destination base (type: string of digits)

1. **Rapid conversion method method**

**d.1 ) Conversion from the source base p=2k , p** € **{4=22,8=23,16=24} into the destination base  2**

Rule:

Each digit from the source number in base *p*=2*k*, will be replaced by the corresponding group of *k* binary digits (adding if it is necessary insignificant zeros to the left).

**d.2 )** **Conversion from base 2 into the destination base q=2*k* ,  q** € **{4=22,8=23,16=24}**

**Rules:**

* from right to left make groups of *k*binary digits (adding to the left insignificant zeros to have a complete group);
* the groups will be replaced by the corresponding digits in base *q*=2*k*