Documentation

**Problem statement:** Write an application that implements algorithms for:

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

**Subalgorithm diagram:**

* arithmetic operations with positive integers

main\_menu()

division\_of\_two\_numbers\_in\_a\_base\_p()

substraction\_of\_two\_numbers\_in\_a\_base\_p()

addition\_of\_two\_numbers\_in\_a\_base\_p()

multiplication\_of\_two\_numbers\_in\_a\_base\_p()

convert\_result\_list\_to\_result\_string()

* conversions of natural numbers

main\_menu()

power\_of\_two()

conversion\_using\_substitution\_method()

conversion\_using\_method\_of\_succesive\_divisions()

conversion\_using\_rapid\_conversions()

**Used data type specification:**

Main data used:

* number, number1, number2 – string representing a number in a given base (operations are performed on the numbers symbolized by these strings)
* base, base1, base2 – integers representing a base (2, 3, 4, .., 16) in which operations will be performed
* result\_list – a list that will contain the digits of the result of an operation (for some operations the digits will be represented in the list in reverse order)
* result\_string – the result\_list will be used to create the result\_string representing the result of an operation
* current\_digit – a character representing the digit from a number which we work with currently

**Pseudocode for important algorithms used:**

* Addition of two numbers in a base p

This algorithm performs the addition of two numbers in a base p like on paper. The digits are taken from right to left, added one by one while keeping in mind the carry. Every digit is added in a list which contains in the end the digits of the result in reverse order.

The function convert\_result\_list\_to\_result\_string handles the list and creates a string that represents the result.

Conditions:

1. number1, number2 – the numbers to be added (represented on strings)
2. length\_of\_first\_number, length\_of\_second\_number – the lenghth of the numbers (represented on int)
3. length\_of\_first\_number >= length\_of\_second\_number
4. result\_list – an empty list

Pseudocode:

while length\_of\_first\_number >= 0:

if length\_of\_second\_number >= 0: (there are still digits to be addes from number2)

set last\_digit\_of\_first\_number\_int as the value represented by the last character in number1 (‘0’ -> 0, ‘1’ -> 1, …, ‘F’ -> 15)

set last\_digit\_of\_second\_number as the value represented by the last character in number2 (‘0’ -> 0, ‘1’ -> 1, …, ‘F’ -> 15)

set sum\_of\_digits to the sum of the two digits and the carry

if\_sum\_of\_digits >= base:

sets carry to 1

substracts from sum of digits the base

endif

else:

sets carry to 0

end else

add into the result list, sum\_of\_digits (a digit of the result)

substract one from length\_of\_first\_number, length\_of\_second\_number

endif

else: (if number2 doesn’t have any digits anymore, but number 1 has)

set last\_digit\_of\_first\_number\_int as the value represented by the last character in number1 (‘0’ -> 0, ‘1’ -> 1, …, ‘F’ -> 15)

set sum\_of\_digits to last\_digit\_of\_first\_number\_int + carry

if\_sum\_of\_digits >= base:

sets carry to 1

substracts from sum of digits the base

endif

else:

sets carry to 0

end else

add into the result list, sum\_of\_digits (a digit of the result)

subtract 1 from length\_of\_first\_number

endif

if carry == 1:

add into the result list the digit 1

* Subtracting two numbers in a base p

This algorithm performs the subtraction of two numbers in a base p like on paper. The digits are taken from right to left, subtracted one by one while keeping in mind the borrow. Every digit is added in a list which contains in the end the digits of the result in reverse order.

The function convert\_result\_list\_to\_result\_string handles the list and creates a string that represents the result.

Conditions:

1. Needed data:
   1. minuend – string
   2. subtrahend – string
   3. base – int (2, 3, ..,16)
   4. length\_of\_minuend, length\_of\_subtrahend – number of characters in minuend and subtrahend, int
   5. result\_list – an empty list
2. minuend > subtrahend

Pseudocode:

while length\_of\_minuend >= 0:

if length\_of\_subtrahend >= 0:

set last\_digit\_of\_minuend to last character in minuend

set last\_digit\_of\_subtrahend to last character in subtrahend

set last\_digit\_of\_minuend\_int to the value that the character represents (‘0’ -> 0, ‘1’ -> 1, …, ‘E’ -> 14, ‘F’ -> 15)

set last\_digit\_of\_subtrahend\_int to the value that the character represents (‘0’ -> 0, ‘1’ -> 1, …, ‘E’ -> 14, ‘F’ -> 15)

if borrow == - 1 (there is a previous borrow):

subtract 1 from last\_digit\_of\_minuend\_int

set borrow to 0

endif

set difference\_of\_digits to last\_digit\_of\_minuend\_in –last\_digit\_of\_subtrahend\_int – borrow

append to the result\_list, difference\_of\_digits

subtract 1 from length\_of\_subtrahend

endif

else:

set last\_digit\_of\_minuend to last character in minuend

set last\_digit\_of\_minuend\_int to the value that the character represents (‘0’ -> 0, ‘1’ -> 1, …, ‘E’ -> 14, ‘F’ -> 15)

if borrow == -1: (there is a previous borrow)

if last\_digit\_of\_minuend\_int < 1:

set borrow to -1

set last\_digit\_of\_minuend\_int to last\_digit\_of\_minuend\_int + base – 1

endif

else:

set last\_digit\_of\_minuend\_int to last\_digit\_of\_minuend\_int – 1

set borrow to 0

end else

endif

append to result\_list, last\_digit\_of\_minuend\_int

end else

set length\_of\_minuend to length\_of\_minuend – 1

end while

* Multiplication of two numbers in a base p

This algorithm performs the multiplication of two numbers in a base p like on paper. The digits of the given number are taken from right to left, multiplied one by one by the given digit while keeping in mind the possible transport. Every digit is added in a list which contains in the end the digits of the result in reverse order.

The function convert\_result\_list\_to\_result\_string handles the list and creates a string that represents the result.

Conditions:

Needed data:

* 1. number1 – string
  2. number2 – string of one character
  3. base – int (2,3,4,..16)
  4. result\_list – empty list
  5. length\_of\_number – the number of characters in number1

Pseudocode:

set number2\_int to the value that the character number2 represents (‘0’ -> 0, ‘1’ -> 1, .., ‘F’ ->15)

set transport to 0

while length\_of\_number >= 0:

set last\_digit\_of\_number\_int to the value that the last character in number1 represents (‘0’ -> 0, ‘1’ -> 1, .., ‘F’ ->15)

set current\_product to transport+number2\_int\*last\_digit\_of\_number\_int

set quotient to current\_product / base

ser remainder to current\_product % base

append to the result\_list the remainder

set transport to quotient

set length\_of\_number to length\_of\_number – 1

end while

if transport != 0:

append to result\_list, transport

endif

* Division of two numbers in a base p

This algorithm performs the division of two numbers in a base p like on paper. The digits of the given number are taken from left to right, divided one by one by the given digit while keeping in mind the possible transport. Every digit is added in a list which contains in the end the digits of the result in the right order. The remainder of the division is the last transport.

The function convert\_result\_list\_to\_result\_string handles the list and creates a string that represents the result.

Conditions:

Needed data:

* 1. dividend – string
  2. divisor – character
  3. base – int (2,3,..,16)
  4. result\_list – empty list
  5. length\_of\_dividend – number of characters in dividend

Pseudocode:

set divisor\_int to the value represented by the character divisor (‘0’ -> 0, ‘1’ -> 1, …, ‘F’ -> 15)

for index from 0 to length\_of\_dividend – 1:

set current\_digit\_int to the value represented by the character at index in dividend (‘0’ -> 0, ‘1’ -> 1, …, ‘F’ -> 15)

set result\_digit to (base \* transport + current\_digit\_int) / divisor\_int

set transport to (base \* transport + current\_digit\_int) % divisor\_int

append to result\_list, current\_digit

end for

* Conversion using substitution method

The algorithm performs the conversion using substitution method. It is performed when base1 (given base) is smaller than base2 (the base we want to have the result in) . The number in the wanted base is obtained by adding one by one the digits of the number multiplied with the given base at the corresponding power (base1^0 – for last digit, base1^1 – for second to last digit, …). All operations are performed in the base in which we convert (base2).

Conditions:

1. Needed data
   1. number – string
   2. base1 – int
   3. base2 – int
   4. length\_of\_number – the number of character in number, int
   5. base1 < base2

Pseudocode:

set power to the character ‘1’

set number\_in\_base2 to the character ‘0’

set base1\_str to the corresponding character of base1 (0->’0’, 1->’1’,…, 15->’F’)

while length\_of\_number >= 0:

set last\_digit to the character at index length\_of\_number in number

if last\_digit != 0:

set second\_operand to the product of power, last\_digit in base2

(multiplication performed using the algorithm from above)

set number\_in\_base2 to the sum of number\_in\_base2, second\_operand in base2

(addition performed using the algorithm from above)

end if

set power to the product of power, base1\_str in base2

set length\_of\_number to length\_of\_number – 1

end while

* The method of successive divisions

The algorithm performs the conversion using the method of successive divisions. It is performed when base1 (given base) is larger than base2 (the base we want to have the result in). The number in the wanted base is obtained by dividing the number by base2, obtaining a quotient and a remainder. The quotient is divided again by base2, obtaining a new quotient and a new reminder. The process of successive divisions ends when 0 is obtained as quotient. The remainder, in the reverse order of obtaining them, are the digits of the new representation in base2.

The remainders are added one by one into a list (result\_list) and the function convert\_result\_list\_to\_result\_string handles the list and creates a string that represents the result.

Conditions:

Needed data:

1. number – string
2. base1 – int (2,3,4,..,16)
3. base2- int (2,3,4,…,16)
4. result\_list – an empty list
5. base1 > base2

Pseudocode:

set base2\_str to the character represented by the value of base2 (2->’2’,3->’3’,…,15->’F’)

while True:

set number, remainder to the values obtained by dividing number to base2\_str (division performed in base1 using the algorithm above, number = number / base2\_str, remainder = number % base2\_str)

append to result\_list the remainder

if number == 0:

break

endif

end while

* Conversion using rapid conversion

The algorithm performs the conversion using the rapid conversions. It is performed when base1 and base2 are power of 2. The number is first of all represented in base 2 (using the method of successive divisions). A group of 2 digits in base 2, represents a digit in base 4. A group of 3 digits in base 2, represents a digit in base 8. A group of 4 digits in base 2 represents a digit in base 16. The number is separated in group of digits (from right to left) and every group is converted into the corresponding digit in the wanted base. If the last group (the leftmost one) does not have enough digits, an extension with 0 is performed. The digits obtained are added one by one in the result\_list and converted in the result\_string which represents the result of the conversion.

Conditions:

Needed data:

1. number – string
2. base1 – int (2, 4, 8, 16)
3. base2 – int (4, 8, 16)
4. length\_of\_number – number of characters in number

Pseudocode:

set number\_of\_elements\_in\_group\_of\_digits to 2 if base2 == 4, 3 if base2 == 8, 4 if base2 == 16

if base1 != 2

set number to number represented in base2 (using the successive divisions algorithm)

end if

while length\_of\_number > 0:

set current\_digit to empty string

if length\_of\_number + 1 < number\_of\_elements\_in\_group\_of\_digits: (not enough digits for the last group)

for index from 0 to number\_of\_elements\_in\_group\_of\_digits – length\_of\_number

(perform the extension with 0)

append to the current\_digit string the character ‘0’

end for

for index from 0 to length\_of\_number – 1

append to current\_digit string the character from number at index (number[index])

end for

end if

else:

for index in range number\_of\_elements\_in\_group\_of\_digits – length\_of\_number + 1 to length\_of\_number + 2

append to current\_digit string the character from number at index (number[index])

end for

end else

set current\_digit\_in\_base\_2 to the representation of current\_digit in base2 (using the conversion using substitution method algorithm from above)

append to result\_list, current\_digit\_in\_base2

set length\_of\_number to length\_of\_number – number\_of\_elements\_in\_group\_of\_digits

Set of test data:

* Addition: 1, 1, 8, 5671, 6542

Expected result: 14433

* Subtraction: 1, 2, 5, 11111, 4444

Expected result: 1112

* Multiplication: 1, 3, 16, 2ABC1, 9

Expected result: 1809C9

* Division: 1, 4, 7, 123456, 5

Expected result: quotient 16224, remainder 0

* Conversion using rapid conversion: 2, 4, 12322, 8

Expected result: 672

* Conversion using substitution method: 2, 5, 1232443, 16

Expected result: 5E3B

* Conversion using the method of successive divisions: 2, 8, 76563, 3

Expected result: 1122001110

* End the program: 3