Convert

project documentation

Ferrante Agustin LU 120314

Islas Agustin LU 128629

UNS 2021

Index

* 1 Requirement definition and specification 2
  + 1.1 General definition of the project 2
  + 1.2 Project requirements specifications 2
    - 1.2.1 General requirements 2
    - 1.2.2 Functional requirements 2
    - 1.2.3 Project authorship 3
    - 1.2.4 System scopes 3
  + 1.3 Procedure specifications 3
    - 1.3.1 Development procedures 3
      * 1.3.1.1 Used tools 3
      * 1.3.1.2 Planification 3
    - 1.3.2 Installation 3
* 2 System architecture 4
* 3 Technical documentation 5
  + 3.1 File reference: convert.c 5
    - 3.1.1 Function documentation 5
    - 3.1.2 Variable documentation 6
  + 3.2 File reference: argumentparser.h/.c 7
    - 3.2.1 Function documentation 7
    - 3.2.2 Variable documentation 9
  + 3.3 File reference: baseconverter.h/.c 11
    - 3.3.1 Function documentation 11

**1 Requirement definition and specification**

**1.1 General definition of the project**

This program was developed with the intent to provide a tool for those who need to work with different number bases. This piece of software computes the conversion of a number between two bases as requested by the user, giving the possibility of visualizing the step-by-step process of conversion.

The conversion is realized as follows:

* For the integer part of the number:
  + From any base, to base 10: Using the multiplication method.
  + From base 10, to other base: Using the division method.
* For the fractional part of the number:
  + From any base, to base 10: using the division method.
  + From base 10, to other base: using the multiplication method.

**1.2 Project requirements specifications**

**1.2.1 General requirements**

* The program was compiled using the GCC flag -Wall, without triggering warnings of any kind.
* Efficient and proper use of dynamic memory is guaranteed.
* The program was carried out using modular programming (including functions and procedures).
* The code is commented and uses representative names, to facilitate its reading.

**1.2.2 Functional requirements**

The program execution syntax is as follows:

**convert -n <number> [-s <source\_base] [-d dest\_base] [-v] [-h]**

* The argument -n is required and is the number to convert.
* If the arguments -s and -d are specified, the conversion of the received number <number> will be carried out, from the origin base <source\_base>, to the destination base <dest\_base>, showing the result on the screen. Arguments can be entered in any order and if the arguments -s or -d are not specified, base 10 will be assumed.
* If the parameter -v is specified, the intermediate computations made during the conversion process will be shown.
* If the parameter -h is specified, the program will only show a help screen, with information about the syntax and semantics necessary to use the program.

**1.2.3 Project authorship**

The authors of this document guarantee the authorship of the software project.

**1.2.4 System scopes**

* The bases supported by the program are those that belong to the range [2, 16].
* The program will not convert numbers whose integer part exceeds 10 digits of magnitude (in the destination or source base), or whose fractional part exceeds 5 digits (in the source base).
* The program allows the conversion of any number, as long as it can be represented in the destination base.
* The program will limit the fractional part of the number if it exceeds 5 digits in length (truncating the outcome).

**1.3 Procedure specifications**

**1.3.1 Development procedures**

**1.3.1.1 Used tools**

* Code::Block IDE
* GNU GCC compiler
* Doxygen

**1.3.1.2 Planification**

1. A general diagram of the program was made.
2. The tasks of the program were specified in different modules.
3. The modules were implemented.
4. The modules were tested in isolation.
5. All the modules of the program were joined.

**1.3.2 Installation**

On Windows OS:

* Extract the compressed files.
* To ease the use of the program, add its directory to the Windows PATH.

**2 System architecture**

The structure of the program is constituted as follows:

There are three modules, the main module of the program, named convert, and two auxiliary modules: argumentparser and baseconverter (*Figure 1*).

* The convert module controls the main flow of the program.
* The argumentparser module is in charge of analizing the program input.
* The baseconverter module is in charge of doing all base changes.

This project uses the following external dependencies: stdio, stdlib, string and math libraries.

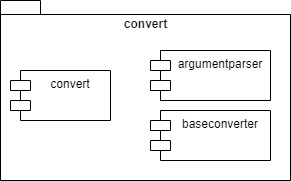
**3 Technical documentation**

Figure 1

**3.1 File reference: convert.c**

#include <stdio.h>  
#include <stdlib.h>  
#include "**argumentparser.h**"  
#include "**baseconverter.h**"

3.1.1 Function documentation

◆ freeAll()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| void freeAll | ( |  | ) |  |

Frees all global variables.

◆ main()

|  |  |  |  |
| --- | --- | --- | --- |
| int main | ( | int | argc, |
|  |  | char \* | argv[] |
|  | ) |  |  |

Program's main function.

**Parameters**

|  |  |
| --- | --- |
| **argc** | Number of arguments. |
| **argv** | Array of strings, the arguments for the execution. |

◆ showHelp()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| void showHelp | ( |  | ) |  |

Prints to the console the convert program help message.

3.1.2 Variable documentation

◆ destArgValue

|  |
| --- |
| int\* destArgValue |

Pointer to integer, the value of the destination base argument.

◆ helpArgPresent

|  |
| --- |
| int\* helpArgPresent |

Pointer to integer, to store 1 if the help argument is present, 0 if not.

◆ numberArgFractionValue

|  |
| --- |
| char\* numberArgFractionValue |

Char array, the value of the fractional part of the number argument.

◆ numberArgIntegerValue

|  |
| --- |
| char\* numberArgIntegerValue |

Char array, the value of the integer part of the number argument.

◆ sourceArgValue

|  |
| --- |
| int\* sourceArgValue |

Pointer to integer, the value of the source base argument.

◆ viewArgPresent

|  |
| --- |
| int\* viewArgPresent |

Pointer to integer, to store 1 if the view argument is present, 0 if not.

**3.2 File reference: argumentparser.h/.c**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>

3.2.1 Function documentation

◆ argToChar()

|  |  |  |  |
| --- | --- | --- | --- |
| void argToChar | ( | char \* | str, |
|  |  | char \* | output |
|  | ) |  |  |

Stores the identifier character for the given string in the given pointer.

The identifier character is the second character of the argument, for the arguments of the form "-x", where x is any character.

**Parameters**

|  |  |
| --- | --- |
| **str** | A string, the argument to check. Valid arguments are "-n", "-s", "-d", "-h", and "-v". |
| **output** | A pointer to char, where the output will be stored. If the input argument is not valid, the output is a null character. |

◆ freeAll()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| void freeAll | ( |  | ) |  |

Frees all global variables.

◆ parseArguments()

|  |  |  |  |
| --- | --- | --- | --- |
| void parseArguments | ( | int \* | argc, |
|  |  | char \* | argv[] |
|  | ) |  |  |

Checks if the given arguments are valid for the program.

**Parameters**

|  |  |
| --- | --- |
| **argc** | The number of arguments. |
| **argv** | An array of strings, the arguments for the program execution. |

◆ parseBase()

|  |  |  |  |
| --- | --- | --- | --- |
| void parseBase | ( | const int \* | base, |
|  |  | int \* | baseArg, |
|  |  | int \* | control |
|  | ) |  |  |

Checks if a given base is valid, and, if it is, stores it in the given pointer.

**Parameters**

|  |  |
| --- | --- |
| **base** | A pointer to integer, the base to check. |
| **baseArg** | A pointer to integer, pointing to the memory address in which, in case of being valid, the base will be stored. |
| **control** | A pointer to integer. This procedure will store EXIT\_FAILURE on it if an error occurred. |

◆ parseN()

|  |  |  |  |
| --- | --- | --- | --- |
| void parseN | ( | char \* | toParseNArg, |
|  |  | int \* | control |
|  | ) |  |  |

Checks if the given array is a valid number in the current source base argument value.

**Parameters**

|  |  |  |
| --- | --- | --- |
| **toParseNArg** | | The array to check. |
| **control** | A pointer to integer. This procedure will store EXIT\_FAILURE on it if an error occurred. | | |

◆ toUpperCase()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| void toUpperCase | ( | char \* | character | ) |  |

Converts the given character to upper case.

A pointer to the character to convert.

◆ validateDigit()

|  |  |  |  |
| --- | --- | --- | --- |
| void validateDigit | ( | const char \* | digit, |
|  |  | const int \* | base, |
|  |  | int \* | control |
|  | ) |  |  |

Checks if the given character is a valid digit for a given base.

**Parameters**

|  |  |
| --- | --- |
| **digit** | A pointer to char, the character to check. |
| **base** | A pointer to integer, pointing to a value in the range [2, 16], the base to check. |
| **control** | A pointer to integer. This procedure will store EXIT\_FAILURE on it if an error occurred. |

**Attention**

Calling this procedure with base outside range [2, 16] is undefined behavior.

3.2.2 Variable documentation

◆ destArgValue

|  |  |  |
| --- | --- | --- |
| |  | | --- | | int\* destArgValue | | extern |

Pointer to integer, the value of the destination base argument.

◆ helpArgPresent

|  |  |  |
| --- | --- | --- |
| |  | | --- | | int\* helpArgPresent | | extern |

Pointer to integer, to store 1 if the help argument is present, 0 if not.

◆ numberArgFractionValue

|  |  |  |
| --- | --- | --- |
| |  | | --- | | char\* numberArgFractionValue | | extern |

Char array, the value of the fractional part of the number argument.

◆ numberArgIntegerValue

|  |  |  |
| --- | --- | --- |
| |  | | --- | | char\* numberArgIntegerValue | | extern |

Char array, the value of the integer part of the number argument.

◆ sourceArgValue

|  |  |  |
| --- | --- | --- |
| |  | | --- | | int\* sourceArgValue | | extern |

Pointer to integer, the value of the source base argument.

◆ viewArgPresent

|  |  |  |
| --- | --- | --- |
| |  | | --- | | int\* viewArgPresent | | extern |

Pointer to integer, to store 1 if the view argument is present, 0 if not.

**3.3 File reference: baseconverter.h/.c**

#include <stdio.h>  
#include <stdlib.h>  
#include <math.h>

3.3.1 Function documentation

◆ digitChar()

|  |  |  |  |
| --- | --- | --- | --- |
| void digitChar | ( | const int \* | num, |
|  |  | char \* | output |
|  | ) |  |  |

Stores the character that represents the given value in the output pointer.

**Parameters**

|  |  |
| --- | --- |
| **num** | A pointer to an integer number between 0 and 15. |
| **output** | A pointer to a char. This is where the output is stored. |

**Warning**

An invalid num entry will cause the output pointer to become NULL.

◆ digitValue()

|  |  |  |  |
| --- | --- | --- | --- |
| void digitValue | ( | const char \* | num, |
|  |  | int \* | output |
|  | ) |  |  |

Stores the value of the given character in the output pointer.

The stored value is value of the character when used to represent numbers in different bases (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F; for 0 to 15 respectively).

**Parameters**

|  |  |
| --- | --- |
| **num** | A pointer to a char between 0 and 9 or between A and F. |
| **output** | A pointer to an integer. This is where the output character is stored. |

**Warning**

An invalid num entry will cause the output pointer to become NULL.

◆ fractionDivisionMethod()

|  |  |  |  |
| --- | --- | --- | --- |
| float \* fractionDivisionMethod | ( | const char \* | nFraction, |
|  |  | const int \* | sourceBase, |
|  |  | const int \* | viewArgument |
|  | ) |  |  |

Calculates a base change using the fractional division method.

Division method for fractional numbers is used to change base operating in the destination base, so it's used to change from any base to base 10.

**Parameters**

|  |  |
| --- | --- |
| **nFraction** | An array of char, representing a number coded in a base from 2 to 16. |
| **sourceBase** | A pointer to integer, with value from 2 to 16, representing the base of the input number. |
| **viewArgument** | A pointer to integer, with value 1 or 0, depending on if the intermediate calculations need to be shown on the console. |

**Returns**

A pointer to a floating-point number, the input number converted to base 10.

**Note**

The returned pointer is allocated dynamically.

◆ fractionMultiplicationMethod()

|  |  |  |  |
| --- | --- | --- | --- |
| char \* fractionMultiplicationMethod | ( | float \* | nFraction, |
|  |  | const int \* | destBase, |
|  |  | const int \* | viewArgument |
|  | ) |  |  |

Calculates a base change using the fractional multiplication method.

Multiplication method for fractional numbers is used to change base operating in the source base, so it's used to change from base 10 to any base.

**Parameters**

|  |  |
| --- | --- |
| **nFraction** | A pointer to a floating-point number, the number to convert. |
| **destBase** | A pointer to integer, with value from 2 to 16, representing the destination base for the conversion. |
| **viewArgument** | A pointer to integer, with value 1 or 0, depending on if the intermediate calculations need to be shown on the console. |

**Returns**

An array of characters, the input number converted to the given base.

**Note**

The returned pointer is allocated dynamically.

**Attention**

The value stored in the address pointed by nFraction is consumed by this function.

◆ integerDivisionMethod()

|  |  |  |  |
| --- | --- | --- | --- |
| char \* integerDivisionMethod | ( | long long \* | nInteger, |
|  |  | const int \* | destBase, |
|  |  | const int \* | viewArgument |
|  | ) |  |  |

Calculates a base change using the integer division method.

Division method for integers is used to change base operating in the source base, so it's used to change from base 10 to any base.

**Parameters**

|  |  |
| --- | --- |
| **nInteger** | A pointer to a 64-bit integer, the base 10 number to convert. |
| **destBase** | A pointer to integer, with value from 2 to 16, representing the destination base for the conversion. |
| **viewArgument** | A pointer to integer, with value 1 or 0, depending on if the intermediate calculations need to be shown on the console. |

**Returns**

An array of characters, the input number converted to the given base.

**Note**

The returned pointer is allocated dynamically.

◆ integerMultiplicationMethod()

|  |  |  |  |
| --- | --- | --- | --- |
| long long \* integerMultiplicationMethod | ( | const char \* | nInteger, |
|  |  | const int \* | sourceBase, |
|  |  | const int \* | viewArgument |
|  | ) |  |  |

Calculates a base change using the integer multiplication method.

Multiplication method for integers is used to change base operating in the destination base, so it's used to change from any base to base 10.

**Parameters**

|  |  |
| --- | --- |
| **nInteger** | An array of char, representing a number coded in a base from 2 to 16. |
| **sourceBase** | A pointer to integer, with value from 2 to 16, representing the base of the input number. |
| **viewArgument** | A pointer to integer, with value 1 or 0, depending on if the intermediate calculations need to be shown on the console. |

**Returns**

A pointer to a 64-bit integer, the input number converted to base 10.

**Note**

The returned pointer is allocated dynamically.

◆ isRepresentable()

|  |  |  |  |
| --- | --- | --- | --- |
| int \* isRepresentable | ( | const long long \* | num, |
|  |  | const int \* | baseDest |
|  | ) |  |  |

Verifies if the given number in base 10 is representable in the given base with a precision of 10.

**Parameters**

|  |  |
| --- | --- |
| **num** | A pointer to a 64-bit integer, the number to check. |
| **baseDest** | A pointer to an integer, the base to check. |

**Returns**

A pointer, pointing to 1 if the number is representable in the given base.

A pointer, pointing to 0 if the number is not representable in the given base.

**Note**

The returned pointer is allocated dynamically.