

JASM CODE COMPARISON WITH OTHER LANGUAGES

Admin @ Jasm Tech . com

Copyright JasmSoft 2020-2025. All rights reserved.

Here we compare Jasm code with several other languages doing the same algorithm. This shows the simplicity of Jasm code (smaller, cleaner, easier to understand).

The algorithm will convert water temperature into Celsius. It will return an error if the temperature is out of range, the units are wrong, or it fails for any other reason.

Algorithm specification:

- The conversion procedure is named “ConvertWaterTemperatureToC”.
- The input is a value and a unit.
- Value: The value is a floating-point temperature in either Celsius or Fahrenheit.
- The acceptable temperature range is 0 through 100 Celsius (floating point).
- Unit: For Celsius it accepts the symbol `CELSIUS` (value 0), the number 0, the string “0”, or the string “C”.
- Unit: For Fahrenheit it accepts the symbol `FAHRENHEIT` (value 1), the number 1, the string “1”, or the string “F”.
- The answer will be returned to the caller in a parameter named “temperatureInC”.
- An integer error code will be returned if conversion is not possible. “No error” will be 0.
- Descriptive text for the error code must be provided in a way simple for the system to get.

Jasm

12 lines of code written by the programmer, very easy to understand.

```
// Water temperature may only be 0 to 100C
typedef float as water_temperature_in_c( 0.0 to 100.0 )
```

```
// Temperature can be in these units
Class TemperatureUnits is Enum:
  Index  Symbol  Name
  0       CELSIUS  "C"
  1       FAHRENHEIT "F"
```

```
Convert water temperature into Celsius.
The answer is returned in temperatureInC.
Returns standard error code.
```

```
proc <jerr> ConvertWaterTemperatureToC (
  temperatureInC is ref to water_temperature_in_c,
  value is float,
  units is TemperatureUnits.sym_type) ; e :
  // Hidden range checking for all arguments here

  if (units == TemperatureUnits.CELSIUS):
    temperatureInC := value
  else // FAHRENHEIT:
    temperatureInC := (value-32) * 5.0 / 9.0
  return ERROR_NONE
```

Note 1: The `water_temperature_in_c` type definition element prevents an invalid water temperature with compile & run time checking. The procedure will return `ERROR_VALUE_OUT_OF_RANGE` if a value is rejected.

Note 2: The `TemperatureUnits` enum in the **black box** converts between numbers, symbols, & strings. It also rejects any values not on its list with a descriptive error.

Note 3: Jasm error codes come with descriptive text and the line number where the error occurred.

Note 4: Comments are blue; The double blue box is a comment.

Note 5: The **red line** is a reference that returns a value.

Note 6: Error checking is never skipped because Jasm does it automatically (preventing bugs). No other language does this. The error checking is not shown here because you don't have to code it.

C/C++

79 lines of user code. 5 lines are boilerplate. 42 lines are for low-level processing; they make the code harder to understand. Overall, this code is **not** easy to understand.

```
// Strip whitespace & carriage returns from a string
// It panics on bad input
static void StripWhitespace(
    char *buffer,           // Pass in a buffer for the output
    size_t bufsize,        // Size of the output buffer
    const char *inputString ) // The input string
{
    assert( buffer != NULL );
    assert( inputString != NULL );
    assert( bufsize > (sizeof(inputString)+1) );
    unsigned int bufferIx = 0;
    unsigned int inputIx = 0;
    buffer[0] = 0;         // initialize output to empty string

    // Find real beginning of input
    while ( isspace(inputString[inputIx])
        && (!iscntrl(inputString[inputIx]))
        && (inputString[inputIx]) )
    {
        if ( (iscntrl(inputString[inputIx]))
            || (!inputString[inputIx]) )
            return;    // Found CR, control char, end of input
        inputIx++;
    }

    // Copy input to output until end is found
    while ( (!iscntrl(inputString[inputIx])
        && (inputString[inputIx])) )
    {
        buffer[bufferIx] = inputString[inputIx];
        buffer[bufferIx+1] = 0; // Terminate the string
        inputIx++;
        bufferIx++;
    }

    // Backtrack in output buffer and kill trailing whitespace
    while ( (isspace(buffer[strlen(buffer)-1]))
        buffer[strlen(buffer)-1] = 0;    // Shorten the string
    }

    // Enumerate the errors we can return
    typedef enum {
        ERROR_NONE,           // ERROR_NONE must be 0
        ERROR_VALUE_OUT_OF_RANGE,
        ERROR_UNKNOWN_UNITS,
        ERROR_LAST           // This is how many there are
    } error_code;
```

```

static const char *GetErrorCodeText(const error_code code)
{
    static const char * const errorCodeText[ERROR_LAST] = {
        "No error",
        "Value out of range"
        "Unknown units"
    };
    if (code < ERROR_LAST)
        return errorCodeText[code];
    return "Unknown error code";
}

// Enumerate the temperature units
typedef enum { CELSIUS, FAHRENHEIT } temperature_unit;
const char * const temperatureUnitNameText[2] = { "C", "F" };
const char * const temperatureUnitNumberText[2] = { "0", "1" };

// Convert water temperature to Celsius using numeric units
// unitsCode==0 means the value is in Celsius
// unitsCode==1 means the value is in Fahrenheit
error_code ConvertWaterTemperatureToC(
    double & temperatureInC,           // Return the answer here
    const double value,                // Temp in unknown units
    const temperature_unit unitsCode ) // 0=C, 1=F
{
    // Units must be within range
    if (unitsCode == CELSIUS)
        temperatureInC = value;
    else if (unitsCode == FAHRENHEIT)
        temperatureInC = (value-32) * 5.0 / 9.0;
    else
        return ERROR_UNKNOWN_UNITS;

    // The value must be within range
    if ( (temperatureInC<0.0) || (temperatureInC>100.0) )
        return ERROR_VALUE_OUT_OF_RANGE;

    // Successful conversation to a water temperature in Celsius
    return ERROR_NONE;
}

// Convert water temperature to Celsius using string units
// Assume code not shown here has stripped of all leading
// and trailing whitespace (but that code must still be written)
error_code ConvertWaterTemperatureToC(
    double & temperatureInC,           // Return the answer here
    const double value,                // Temp in unknown units
    const char * const unitsText )     // Text for the units
{
    // Check to see if unitsText is anything we recognize

```

```
const int celsius = strcmp( unitsText, "C" );
const int zero = strcmp( unitsText, "0" );
const int fahrenheit = strcmp( unitsText, "F" );
const int one = strcmp( unitsText, "1" );

// Call the numeric routine matching the units we found
if ( (celsius==0) || (zero==0) )
    return ConvertWaterTemperatureToC( temperatureInC, value, 0 );
if ( (fahrenheit==0) || (one==0) )
    return ConvertWaterTemperatureToC( temperatureInC, value, 1 );

// The units didn't match if we got here
return ERROR_UNKNOWN_UNITS;
}
```

Note 1: Returning line number & file information is possible in C, but would add more complexity. We did not do it here.

Note 2: Code written in C++ needs a lot of hand-written low-level routines. Examples below include `GetErrorCodeText` and `StripWhitespace`. The latter is needed to strip all carriage returns and whitespace from the input.

Note 3: This code has some error checking. Many programmers skimp on these checks because they have to do it manually. Skimping on error checking is a common source of bugs. Jasm does this error checking automatically.

Note 4: C++ is usually hard to read. It takes some effort to make it more readable.

Python

53 lines of code, 37 of which are for a necessary test program. The code is moderately easy to understand, but the test program is not.

```
def ConvertWaterTemperatureToC( value, units ):
    """Convert a temperature to degrees C.
    value must be 0-100C or 32-212F
    units must be "C", "F", "0", "1", 0, or 1.
    Returns a tuple: (error code, error string,
    temperature in C)."""

    # Put arguments into a known format
    value = float(value)      # force value to number
    units = str(units)       # force units to be string
    units = units.upper()    # convert to uppercase
    units = units.strip()    # remove whitespace
    if units == '0':
        units = 'C'         # change '0' to 'C'
    if units == '1':
        units = 'F'         # change '1' to 'F'

    # value is now a floating-point number (or we crashed)
    # units is now a string: 'C', 'F', or other
    temperatureInC = value   # Start by assuming Celsius
    if units == 'F':
        # Convert Fahrenheit to Celsius
        temperatureInC = (temperatureInC-32) * 5.0 / 9.0
    elif not units == 'C':
        # Error: unknown units are not C or F
        return (2, 'Unknown units', 0.0)

    # The temperature is now in Celsius
    # Range check it
    if (temperatureInC < 0) or (temperatureInC > 100):
        # Error: Temperature is out of range
        return (1, 'Temperature out of range', temperatureInC)

    # The temperature is good. Return it.
    return (0, "No error", temperatureInC)

def UnitTest():
    """Test ConvertWaterTemperatureToC.
    Return True if good, False if bug found."""

    # Tables of test values
    celsiusTuple = ( -1, 0, 50, 100, 101 )
    fahrenheitTuple = ( 31, 32, 122, 212, 213 )
    valueGood = ( False, True, True, True, False )
```

```

# Tables of test units
unitTuple = ( "C", 0, "F", 1, "X", 2 )
unitGood = ( True, True, True, True, False, False )

# First test Celsius
for valueIndex in xrange(5):
    for unitIndex in xrange(6):
        value = celsiusTuple[valueIndex]
        unit = unitTuple[unitIndex]
        expectedGood = valueGood[valueIndex] &
            unitGood[unitIndex]
        resultCode, resultMsg, resultVal =
            ConvertWaterTemperatureToC( value, unit )
        resultGood = (resultCode == 0)
        if not resultGood == expectedGood:
            # Error: result code was wrong
            return False
        if resultGood:
            if not celsiusTuple[valueIndex]
                == resultVal:
                # Error: wrong temperature
                return False

```

```

# Now test Fahrenheit
for valueIndex in xrange(5):
    for unitIndex in xrange(6):
        value = fahrenheitTuple[valueIndex]
        unit = unitTuple[unitIndex]
        expectedGood = valueGood[valueIndex] &
            unitGood[unitIndex]
        resultCode, resultMsg, resultVal =
            ConvertWaterTemperatureToC( value, unit )
        resultGood = (resultCode == 0)
        if not resultGood == expectedGood:
            # Error: result code was wrong
            return False
        if resultGood:
            if not celsiusTuple[valueIndex]
                == resultVal:
                # Error: wrong temperature
                return False

# No errors found
return True

```

Note 1: Python is notorious for being buggy and having latent syntax errors. The usual solution is to write more test code than the code to be tested. The first routine here is the temperature conversion routine and the second is its test program.

Note 2: The test code more than doubles the code to write and bugs to be found. If the programmer skimps on test code, python code will be full of bugs, some of them crash bugs that may not manifest until much later. Jasm's automatic error checking greatly reduces the need for test code.

Note 3: Python does not have constants, enumerations, and so forth. Making systemic changes can be a problem in their absence. Also, there are no symbols like CELSIUS or FAHRENHEIT available for units. Adding, deleting, or changing a unit would be error-prone in code like this; it would be difficult to get them all.