International Journal of Scientific & Engineering Research Volume 8, Issue 5, May-2017
ISSN 2229-5518

Exception Based Scanning and Assigning of Data Variables

Dr. Kumudavalli M.V, Rohit Kumar Singh, Amthul Hai

Abstract— Data validation is a necessity when it comes to data parsing. Choosing whether the given data can fit into the given data type or not is a concern to the programmers. Error in input and ignoring the resulting exceptions may result in undesired working of the program. This is of concern to programmers which takes a lot of their coding time and effort. To end this the current function/class read_var is being developed which checks for the given input, compares the input to match with the given data type and gives exceptions if the data is not suitable and assigns the value if it fits the data type. read_var is a single class with multiple functionalities which allows the programmers to concentrate on their program rather than fixing the problems related to scanning/parsing.

Key words - read_var, data variable, assigning, input stream, Programming Languages

1 Introduction

CONSIDER a signed int variable, it is stored in the memory with 32 bits of space. The first bit is for the sign. The rest would be 31 bits for storing the integer value. Therefore the maximum value is (2^{31}) - 1 or the minimum value is-(2^{31}) + 1. Entering a value beyond this will let the variable store junk value. Now, consider float, it holds fractional values. IEEE 754 is the standard of holding a float value that is 4 bytes and can hold values between 2e38 and 2e-38, but the precision isn't fixed. Finding value of $2^{-20}$and still holding the precision of 20 digits beyond the decimal point is possible whereas float has a standard predefined possible precision up to 6 digits. The precision is not fixed for all the value. This is because only 23 bits allowed for storing fractional value in float. If we had to store the value 7.0/3.0 in a floating point variable the variable would hold the data with only 6 digits of precision.
Since the mantissa is rounded off to 23 bits. 2^{-20} holds exactly 1 in the 20^{th} bit of mantissa making it precise but, converting 7.0/3.0 to binary causes the binary value to be rounded off [3].

2 DATA TYPE ANALYSIS

The data is classified into three types viz., numbers, strings and Boolean. Various other types such as date, time, and currency etc., are data structures based on those three fundamental basic data types [4]. These three types that are further classified into fundamental data types are as shown in Fig 1. Integer data type has 8 bit, 16 bit, 32 bit, 64 and 128 bit (not every architecture supports this type) memory space allocation. Short being 8 bit, int being 16 or 32 bit, long being 32 or 64 bit and long long being 64. They have both unsigned and signed type. The floating point types are float (32 bits), double (64 bits), long double (80 bits).
The Character data types are char (8 bits) and wchar_t(16 bits). They both can be signed or unsigned

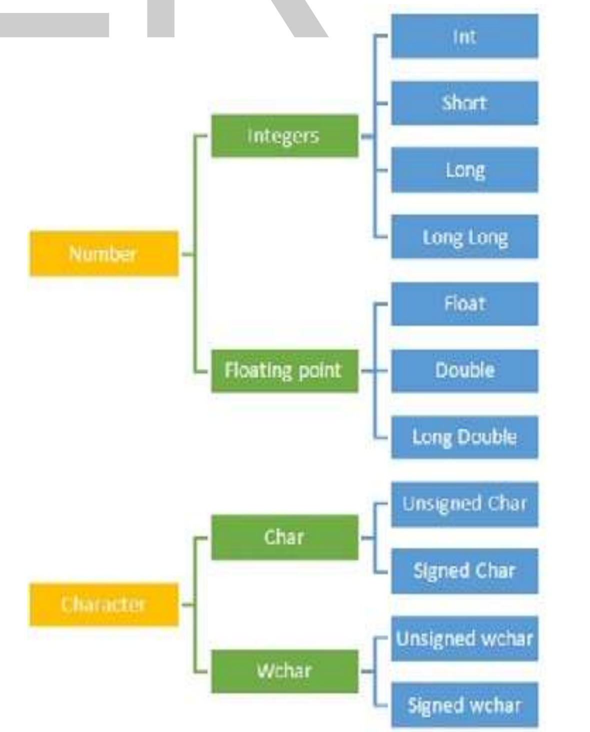


Fig. 1. Description of Fundamental Data Types.

Dr. Kumudavalli M.V, is working as an Assistant Professor at Department of Computer Applications, DSCASC, kumudamanju@gmail.com

Rohit Kumar Singh, is currently pursuing BCA at Department of Computer Applications, DSCASC, rks0nax@gmail.com

Ms. Amthul Hai, is working as an Assistant Professor at Department of Computer Applications, DSCASC, hai.4bu@gmail.com

3 UNDERSTANDING AND ADDRESSING THE PROBLEM

The problems that lie in assigning the values are:
(1) Loss of precision storing data in floating points.
(2) Assigning a value larger than what a given data type can hold.
(3) Invalid data type conversion.
(4) Losing sign value in unsigned variables

The following class with its functions gives a better remedy for the above said problems [5].

A. Function for Non-Pointer data type input

```
template<class generic>
generic variable_in(generic in, int _buffer=64);
```

The above function takes a variable of type generic and has the maximum buffer length. The variable in is the input and variable _buffer is the maximum length of the string. The default value is 64. This function is the entry point for data from input stream.

B. Function for Pointer data type input

```
template<class generic>
generic *variable_in(generic *in, int _buffer=64);
```

The above function will hold same code but redefined declaration for the pointer input.

C. Function for Non-Pointer data assignment

```
template <class generic, class generic2>
generic variable_assign(generic in, generic _in);
```

The above function is the entry point for assigning the variable from one variable into in.The variable
_in is the variable with the data and in is the variable where the data will be stored. Pointer type parameters are used for pointer type assignment.

4 PROGRESSIVE SEQUENCE OF FUNCTIONS FOR ASSIGNMENT AND INPUT

The variable_in and other similar functions are declared in the class named read_var. A new instance of the class has to be created to use the function. For example:

```
read_var temp;   //creating instance of read_var class
temp_var = temp.variable_in(temp_var);
//taking the input
```

The function starts by clearing all the exception and error flags. It assigns all the flag and error codes to NULL. It then checks for the data type of the variable. The function calls check_type(in) which uses the typeid operator to check the type and store it into a character array. It works as below:

```
Char type_code[3];
type_code[0] holds pointer information.
type_code[1] holds sign information.
type_code[2] holds the data type code.
```

The above statements are explained with example below:

```
type_code = new char[3];
//char types.
If(typeid(char) == typeid(in))
        type_code = "001";
if(typeid(unsigned char)==typeid(in)
type_code="011";

//int types
…………
//and so on
```

The next step of the function is to take the input from the input stream and store it in a wchar_t variable. The variable is scanned with the buffer length passed in the argument. The remaining buffer stream is cleared.

The wide string input function for data scanning:

```
//_in is stored in the wide string _cpyvar
_cpyvar = new wchar_t[_buffer];
do {
std::wcin.getline(_cpyvar, _buffer + 1);
//The data to be input
while(std::wcin)
std::wcin.clear();
//ignore remaining characters
}while (!(wcscmp(_cpyvar, L"")));
_cpylen = static_cast<int> (std::wcslen(_cpyvar));
```

The function then proceeds to theData_check() Function:
The data_check() function handles the validation of the variables. It starts by setting the variables is_number, is_pointer, is_unsigned to 0. According to the respective type_code, the function executes the code. Assigns 0 to no error, else assigns respective error code. Error codes are defined as macro in the pre-processor directive region such as#define ec_not_num 0, #define ec_out_of_range 1 etcetera.

```
//Example statement
error_code = 0   //setting error code
if(!num_check(_cpyvar))
{error_code = ec_not_num; return;}
//flag codes are declared as fc_flagname
```

Similarly, the functions are called based on the type of data passed to the parameter. The functions perform validation of the data types. Negligible error such as loss of precision can be set as flags. Errors such as not a numerical value, Invalid format, sign error are considered as errors and cause the function to terminate with the respective error code.

The function proceeds by calling the function show_errors(), depending on user preference, the errors can be shown during run time or ignored. To get the error_code, users can call the function get_error_code or call get_flags to get the flag codes. A user can set the runtime error display by setting the variable show_error_onscreen to false.
If there are no errors, _cpyvar is passed to a temporary widechar stringstream. The stream is pushed to a variable of generic type and assigned to _in.

The above functions and its flow are explained pictorially as in Fig 2.

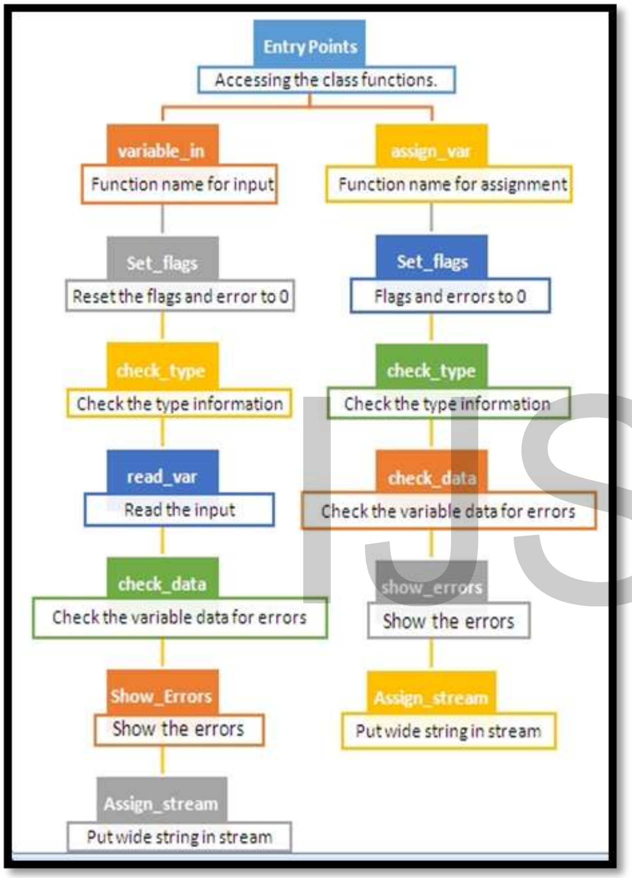


Fig 2. Flow of Functions

5. EXPERIMENTS AND RESULTS

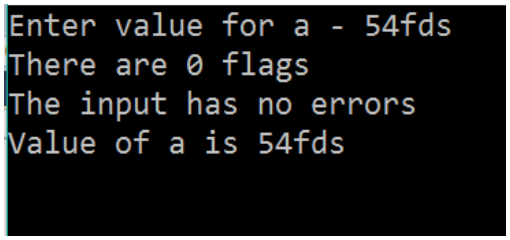
Few C++ Programs with the author developed functions are given below:

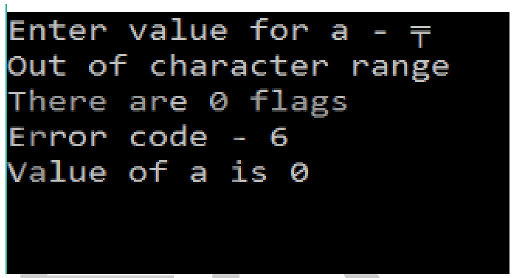
1. Program with CharacterVariables

```
#include "h_read_var.h"        //declaring the header file for read_var_class
int main()
{
    char *a;
    read_var x;        //author's defined class
    std::cout << "Enter value for a - ";
    a = x.variable_in(a);
    //author's defined function
    std::cout << "There are " <<x.get_flag();std::cout << " flags" << std::endl;
    if (x.get_error_code() == 0)

std::cout << "The input has no errors";
    std::cout<< std::endl <<"Value of a is "<<a;
        else std::cout << "Error code - "
    std::cout<<x.get_error_code();
    return 0;
}
```

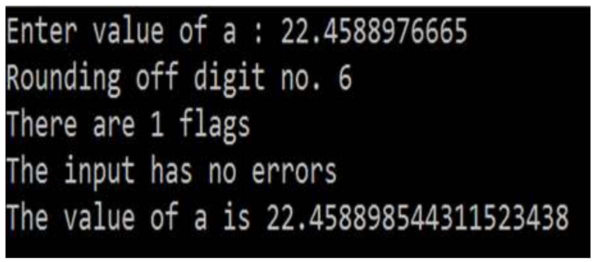
Outputs:





2. Program with Float Variables
Replacing char *a with float a

Output:



3. Program with Integer Variables
Replacing char *a with int a;

Output:

```
Enter value for a : 53634634634
Value for int is greater than what it can hold
There are 1 flags
Error code - 7
Value of a is 0
```

Using assign_var to assign value of long a to short b
Output:

```
Enter value for a : 67476
There are 0 flags
The input has no errors
Assigning values to b
Value for short is greater than what it can hold
Error code - 8
Value of a is 67476
Value of b is 0
```

4. Program with Unsigned Long Variables
Replacingchar *a with unsigned long a

Output:

```
Enter value for a : -25
Deprecating the sign
There are 1 flags
The input has no errors
Value of a is 25
```


6. CONCLUSION

Data variables are essential for any programming language. Data parsing, storing data in a data structure is the next step of data analysis which requires data to be an integral part of these fundamental types. To parse the data, there should be better scanning methods that checks or filters the given input and handles the insignificant data. C++ has limited functionalities to such applications and it is very much required for the language to have a feature that makes the requirement of data checking negligible. The read_var class implemented in this paper caters to the needs of these integral checks making it less effortful and hassle-free for a programmer to write codes for such exceptions. It analyzes the data, checks for major errors and minor exceptions that make the program run differently than expected. The read_var class is an essential bit of code which is required for every type of program which requires user intervention in terms of input stream and complicated data structures. With minimum changes the read_var class can be implemented in many other programming languages.

ACKNOWLEDGMENT

The authors thank the Management of Dayananda Sagar Institutions for their encouragement.

REFEENCES

[1] Gunter C.and Mitchell J.,"Theoretical Aspects of Object-Oriented Programming", MIT Press, 1994.

[2] Xiaoyuan Zhou, "Research on teaching of Java Exception Handling", I.J. Education and Management Engineering, 9, 1-7 2012.

[3] Byunghun Lee, Student Member, IEEE, Dae-KyooKim, Senior Member, IEEE, Hyosik Yang, Hyuksoo Jang, Dae seung Hong, and Herbert Falk, "Unifying Data Types of IEC 61850 and CIM", IEEE Transactions on Power Sys tems, VOL. 30, NO. 1, January 2015

[4] Luca Cardelli, Peter Wegner, "On understanding types, data abstraction, and polymorphism", ACM Computing Surveys, Volume: 17, Issue: 4, Pages: 471-523, 1985.

[5] Hanus M., "The integration of functions into logic programming: From theory to practice", Journal of Logic Programming,19 and 20, 583-628, 1994.