International Journal of Research in Science and Technology http://www.ijrst.com
(IJRST) 2014, Vol. No. 4, Issue No. I, January-March **ISSN: 2249-0604**

INCREASING COMPILER INTERACTION IN PROGRAMMING LANGUAGE –C++OX

Shruti Sandal

Research Scholar (Computer Science) JJT University

ABSTRACT

In this paper we describe variable and expression declaration, declaration of range variable (array etc.) in c++ox. These all concepts are defined in c++ but in this we face some problems. Like when we are declaring any variable then we have to mansion data type in front of the variable unless it not properly read by compiler. Second in array if we wants to store and retrieve the value then we can access it from its index value in other word we have to mansion array's index value by iterative statement. In this paper we show that these above concept is run in easy way and remove all the problem in c++ox.

Keywords*:-auto, decltype, someint, Lambda function, Range loop.*

INTRODUCTION

C++ox have some features from which it override the c++.In standard C++ (and C), the type of a variable must be explicitly specified in order to use it. However, with the advent of template types and template metaprogramming techniques, the type of something, particularly the well-defined return value of a function, may not be easily expressed but in c++ox it express in very easy way. It consist auto and decltype keyword for these types of problem and Range I use for reference variables. All these concept describe in detail in this paper. [1]

DETERMINATION OF DATA TYPE

Variables can be declared in any language by following syntax:
-Data type variable name;
In above expression each thing must be mansion without giving the data type of variable it is not recognize its type .So it is necessary in C++ i.e. explicitly define the type. But in C++ox there are two way to define.
Auto
Decltype

1 Auto: - First, the definition of a variable with an explicit initialization can use the auto keyword. It is automatically convert into suitable data type according to the compiler of it .In other word user is not bother for type of variables .ex:
Auto some strange type = boost::bind (&Some Function, _2, _1, some Object);auto other variable = 5;

In above example its result comes into integer. It shows the implicit type convergence. The type of some Strange Type is simply whatever the particular template function override of boost::bind returns for those particular arguments. This is easily known to the compiler, but is not easy for the user to determine upon inspection.

The type of other Variable is also well-defined, but it is easier for the user to determine. It is an int, which is the same type as the integer literal. It is used for reducing the verbosity of programming code.

2 Decltype: The keyword, decltype can be used to compile-time determines the type of an expression. This keyword is more useful as compare to auto because auto is only familiar to compiler but decltype can also be very useful for expressions in code that makes heavy use of operator overloading and specialized types [6]. For example:

```
int someInt;
decltype (someInt) other Integer Variable = 5;
```

In above example whole expression is automatically taken its type [2]. There is no need to specify the data type of every variable.

USE OF LOOP IN C++OX

C^{++} defines a Range in its library. Range means represent a controlled list, much like a container, between two points in that list. These containers are super set of Range. However, the utility of range concepts is such that $C^{++}0x$ will provide a language feature built around them.

The statement for will allow for easy iteration over a range concept:

```
int my_array [5] = {1, 2, 3, 4, 5};
For (int &x: my_array)
    {
      X *= 2;
    }
```

In above example there is no need to give the index number of variable to access the value of variable. Simpliy define it as above and get the appropriate result. The second section, after the ":", represents the range concept being iterated over. In this case, there is a concept map that allows C-style arrays to be converted into range concepts [3]. This could have been a std::vector, or any object that conforms to a range concept. In C^{++} we wants to access the value in this mode then to have to give its index number with the help of iterative statement every time .so this is one of the advantage of $c^{++}ox$ over c^{++}.

APPLICABILITY OF LAMBDA FUNCTION IN C++0X

In previous versions of c++ox library algorithms function such as sorting and searching, define predicate functions near the invocation of the algorithm function call. The language has only one mechanism for this: the ability to define a class inside of a function. c++ is not allowed for it but c++ox have alternative for it i.e. lambda function in it.

The syntax of lambda function is as follow:
[] (int x, int y) {return x + y}

In above returning value do not have name this is only possible when the expression is in the form of lambda function. In other words in the form of return expression. The return type can be omitted entirely if the lambda function's return type is void. The set of variable which is defined in same scope is known as closure [9]. It is defined as follow:-

```
std::vector<int> someList;
int total1 = 0;
std::for_each (someList.begin (),
someList.end (), [&total1] (int x)
{ total1 += x
    };
std::cout << total1;
```

In this total1 is the lambda function,,s closure. It can change its value and its value is stored as stack. Closure variables for stack variables can also be defined without the reference symbol &, which indicates that the lambda function will copy the value. It is possible to use all available stack variables without having to explicitly reference them:

```
std::vector<int> someList;
int total1 = 0;
std::for_each (someList.begin (), someList.end (), [&] (int x)
  { Total1 += x
});
```

According to the above, the lambda function will store the actual stack pointer of the function it is created in, rather than individual references to stack variables.

If [=] is used instead of [&], all referenced variables will be copied, allowing the lambda function to be used after the end of the lifetime of the original variables [7].

The default value and this pointer can also be handled by lambda function. Lambda function is only compiler dependent type and its type is only available for compiler. If it is used as a function parameter then type must be as template.

CONCLUSION

In this paper we show that how c++ox make more user friendly environment. Type determination is automatically in this. There are no need to specify explicitly it with every variable which is clearly show in this paper. Also describe the new way of specifying the reference variables (array etc.) and lambda function. In last we can say that this language (c++ox) is new phase of more user friendly age .In other word compiler play a very important role in programming because most of concept is built in the compiler, user is only accessing itsfeature.

REFERENCES

[1] *Alexandrescu, Andrei; Herb Sutter (2004). C++ Design and Coding Standards: Rules and uidelines for Writing Programs. Addison-Wesley. ISBN 0-321-11358-6.*

[2] *[Andersen 04] Andersen, et al. "Preliminary System Dynamics Maps of the Insider Cyber-threat Problem." Proceedings of the 22nd International Conference of the System Dynamics Society. Oxford, England, July 25-29, 2004. Albany, NY: System Dynamics Society, 2004. http://www.cert.org/archive/pdf/InsiderThreatSystemDynamics.pdf.*

[3] *[ANSI 89] American National Standards Institute. American National Standard for Information Systems—Programming Language C (X3.159-1989). Washington, D.C., 1989.*

[4] *[Antill 04] Antill, James. Vstr documentation -- overview. http://www.and.org/vstr.*

[5] *Becker, Pete (2006). The C++ Standard Library Extensions: A Tutorial andReference. Addison-Wesley. ISBN 0-321-41299-0.*

[6] *J. Järvi, B. Stroustrup, D. Gregor, J. Siek, G. Dos Reis (September 12, 2004) Doc No:N1705 Decltype (and auto)*

[7] *[Koenig, Andrew; Barbara E. Moo (2000). Accelerated C++ - Practical Programmingby Example. Addison-Wesley. ISBN 0-201-70353-X.*

[8] *Lois Goldthwaite (October 5, 2007) Doc No: N2437 Explicit Conversion Operators*

[9] *V Samko; J Willcock, J Järvi, D Gregor, A Lumsdaine (February 26, 2006) Doc No:N1968 Lambda expressions and closures for C++*