An Industry Standard Language for DDC Systems: A Look at the "C" Language

J.D. Petze **S. Chaudry**
ASHRAE Associate Member

ABSTRACT

Continued development and acceptance of direct digital control (DDC) systems, coupled with growing sophistication of end users, has resulted in a call for standards in the DDC industry. One result is the initiation of research into the development of standard communication protocols for controller-to-controller and controller-to-host communications. Another development is interest in a standard programming language for these systems.

This paper looks at the issue of a standardized programming language, and analyzes the feasibility of using the well-known programming language "C" as the standard. Because of its history of use in the development of computer software, it is often assumed that "C" is difficult to work with. This paper shows how understandable and English-like "C" can be when used for building control applications, i.e., start/stop control sequencing, interlocking, proportional-integral-derivative control of modulating equipment, etc. The "C" language also lends itself to the easy development of functions to accomplish commonplace or repetitive tasks and provides a looping function that allows multiple sets of points to be assigned to a single program. In addition, the primary commands and instructions of the language can be easily redefined to match the jargon of building control.

THE ISSUE OF STANDARDIZATION

Continued acceptance of direct digital control (DDC) as the optimum method of accomplishing automatic temperature control and energy management in buildings of all types has exerted a maturing influence on DDC manufacturers and users. One of the results is an inevitable desire for more "commonality" or standardization among various manufacturers' products. The most notable evidence of the interest in standards is the current activity surrounding the adoption of an "open" communications protocol for controller-to-controller and host-to-controller communications (ASHRAE Committee, 135 p). Another potential area for standardization among DDC systems is that of the programming language used to create control algorithms. While much of the activity on the issue of standards is led by large users of DDC systems, i.e., universities, real estate management firms, and other corporations responsible for a large number of buildings, a standardized programming language would offer benefits to all users, large and small.

Two Important Notes

Before proceeding further it is important to touch on two important items. The issue of a standardized programming language applies primarily to fully user-programmable systems capable of complete direct digital control of the building environment and implementation of custom control strategies unique to the individual installation. Also, it is important to note that we are referring to standardization of the programming language used to develop the actual control strategies executed by the DDC system—the instructions that operate the building equipment. These algorithms are written by the application programmer. The operator interface for daily building operation is a different area of interaction and would not be affected. Manufacturers would still be able to focus their competitive activities on development of new operator interface tools—the means by which setpoints, schedules, or other parameters are changed, alarms acknowledged, and other manual intervention accomplished. This scenario, a standard language used by various manufacturers to provide differing operator environments, closely parallels the way languages such as C, Pascal, Cobol, and others are used in the computer software market to develop competing products for word processing, spreadsheet analysis, graphics, and CAD applications. The presence of widely accepted language standards has been essential to the growth of this market.

BENEFITS OF A STANDARD PROGRAMMING LANGUAGE

A number of significant benefits can be offered by a standardized programming language. These benefits can be placed in two categories: the learning curve and compatibility of program structure.

The Learning Curve

As technology advances, we all are affected by the information explosion— too much to learn, too much to

J.D. Petze, C.E.M., Vice President of Products and Services for Teletrol Systems, Inc., Manchester, NH; **S. Chaudry,** Senior Systems Engineer, Teletrol Systems Inc., Manchester, NH.

know. The operations staff of a building and the controls staff of a mechanical contracting firm are no exception. One of the obvious benefits of a standardized language is a significant reduction in the time it would take to become familiar with systems from different manufacturers. The differences would be reduced to the areas of hardware, installation, and user interface for daily operator activities.

Use of a standard language would also mean that training would be available from more than one source resulting in an increase in the availability of experienced programmers. This, in turn, would mean that the impact of employee turnover would be reduced for both the contractor installing DDC systems and the facilities staff operating them.

Compatibility Of Program Structure

Another, more subtle advantage offered by a standardized programming language would be the compatibility of programs from installation to installation regardless of manufacturer selected. As an example, assume that you are a building manager with a specific piece of equipment to be controlled, for example, an air handler. The algorithms to be performed are clearly defined in accordance with the type of air handler and the environmental requirements of the facility. They have no bearing on the specific DDC manufacturer chosen (assuming, of course, that the equipment chosen provides the programming flexibility to implement the desired strategies). The program instructions to accomplish the specified control strategies, however, can be radically different due to the difference in the way logic constructs are used by various languages. This presents a problem for the person trying to read and understand a program as well as for the programmer.

EXAMPLES OF LOGIC CONSTRUCT DIFFERENCES

Some languages utilize an IF/THEN/GO TO format: If X is true then go to a specified line of the program to perform a specified instruction.

Another language uses a THEN/IF/GO TO format: Do A, then if X is true go to a specified line to perform a specified instruction.

Others use an IF/THEN/ELSE format:
If X is true do A, otherwise do B.

Although subtle, these different formats affect the structure of a control program. Another significant difference among languages is the difference between "fall-through" execution and "one-line-at-a-time" execution.

In a "fall-through" (FT) type language every line of the control program is executed on every program scan. "One-line-at-a-time" (OLAT) means that only one line of an individual control program is executed on every program scan. There are advantages and disadvantages to both. Speed of decision making is almost always faster with FT languages; a complex logical decision can be made in one program scan. Sequential action algorithms, on the other hand, are typically easier to implement with OLAT languages. The execution of the language lends itself to a "do this then wait until next time" sequence of events.

The relative merits of the two are not the issue here; successful systems of both types are offered by different manufacturers. The point here is that two very different programming approaches are required to accomplish the same strategy using the two different types of languages. The following examples of a simple thermostat control algorithm will demonstrate the formation of a program using both fall-through and one-line-at-a-time execution.

Program for Simple Thermostat Control Using One Line At A Time Execution

```
LINE 1
    OFF, FAN, HEATING, COOLING
    GO TO 2 IF TEMP < 68.0
    GO TO 3 IF TEMP > 76.0
LINE 2
    OFF, COOLING
    ON, FAN, HEATING
    GO TO 1 IF TEMP > 70.0
LINE 3
    OFF, HEATING
    ON, FAN, COOLING
    GO TO 1 IF TEMP < 74.0
```

In this program each line represents one of the three possible control states. Line 1 is the off position. No heating, cooling, or fan operation is provided in this state. Line 2 is heating mode, and Line 3 is cooling mode. Control of the outputs fan, heating, and cooling moves from line to line based on the value of the input temperature. Only one line is executed on any one program scan. The following program accomplishes the same strategy using a fall-through language.

Program for Simple Thermostat Control Using Fall Through Execution:

```
/* Heating Mode */
IF TEMP BELOW 68.0 THEN
    TURN (COOLING, OFF);
    TURN (HEATING, ON);
END_IF
/* Cooling Mode */
IF TEMP ABOVE 76.0 THEN
    TURN (HEATING, OFF);
    TURN (COOLING, ON);
END_IF
/* End Heating Mode When Temp is Satisfied */
IF HEATING IS ON AND TEMP ABOVE 70.0 THEN
    TURN (HEATING, OFF);
END_IF
/* End Cooling Mode When Temp is Satisfied */
IF COOLING IS ON AND TEMP BELOW 74.0 THEN
    TURN (COOLING, OFF);
END_IF
/* Turn On fan When Heating or Cooling is On */
IF HEATING IS ON OR COOLING IS ON THEN
    TURN (FAN, ON);
ELSE
    TURN (FAN, OFF);
END_IF
```

In this program each "IF" statement represents one of the possible states of the program. All of these "IF" statements are evaluated when the entire program is executed each scan. Only the true "IF" statement(s) will be executed.

Virtually all high-level programming languages accepted in the software marketplace today utilize the fall-through execution format. Most programmers are familiar with this type of language format.

OVERVIEW OF MAJOR PROGRAMMING LANGUAGES

The following is an overview of some of the languages that have either an historical significance or an important application in the microcomputer environment.

FORTRAN: This was the first high-level language. It is a popular scientific language because of its facility for number crunching and good array-handling features.

COBOL: This is one of the most widely implemented computer languages. Its primary use is in business applications. COBOL is intrinsically self-documenting because of its English-like syntax.

BASIC: Originally created to teach programming to students, it is still extensively used for educational purposes. BASIC is easy to understand and learn and thus is widely used for small business applications.

PASCAL: Like BASIC, this language was created to teach students the art of programming. Pascal produces highly structured programs that are easy to follow and maintain. It is extensively used on microcomputers.

LISP: The list data type is the basic element of this language. LISP is used for many artificial-intelligence applications.

ALGOL: This was the first block-oriented language. It is used mainly for mathematical problem-solving.

PROLOG: This is a logic-oriented language and is used primarily for artificial-intelligence applications.

MODULA-2: This is an enhanced version of Pascal. It is a multiprocessing language in that coroutines may be executed simultaneously.

RPG: This is a nonprocedural language used mainly for producing business reports.

ADA: This is based on Pascal and was developed primarily for the Defense Department's weapons systems tracking. It is not used for many other applications throughout the federal government.

SMALLTALK: This is an object-oriented language.

High Level/Low Level

Programming languages are often classified as "low level" or "high level." Assembly language, for example, is classified as a low-level language because it translates directly into the machine code that the processor interprets. High-level languages are characterized by compilers that translate the instructions written by the programmer into machine-readable code. They also typically include built-in functions that perform routine tasks such as reading and writing of data to the screen or the printer. In this respect C can be considered a low-level language because it does not include these types of functions. This, however, is not a disadvantage because C provides the flexibility to build these functions based on particular needs.

Of all of these languages C is the most widely used and it is the authors' opinion that it is the most appropriate for building control applications.

"C" AS A STANDARD LANGUAGE

A Brief History

C is a general-purpose language that features economy of expression, modern control flow and data structures, and a rich set of operators. C is not specialized to any particular area of application. Its absence of restrictions and its generality, however, make it more convenient and effective for many tasks than supposedly more powerful languages that are highly specialized for specific applications.

C was designed by Dennis Ritchie in 1972 (Kernighan and Ritchie 1978). It was originally designed for systems programming—that is, for writing programs like compilers, operating systems, and text editors. But it has proven quite satisfactory for other applications as well, including database systems, telephone-switching systems, numerical analysis, engineering programs, and a great deal of text-processing software. Today, C is one of the most widely used languages in the world, and C compilers exist for almost every computer (Kernighan and Ritchie 1988).

Standardization of C Itself

With so large an application base in the computer industry, the compatibility of the various C compilers on the market is an important issue. In 1983 the American National Standards Institute (ANSI) established a committee to provide a comprehensive specification for C. The ANSI standard that defines what is known as ANSI C is expected to be approved in 1988. C compilers on the market support most of the features of this standard.

The Advantages Of "C"—How Does It Address The Standardization Issues?

General Suitability. Any potential standard language must possess certain essential features in order to be able to implement the strategies required in DDC applications. It will need math capability (preferably floating point math with addition, subtraction, multiplication, division, and square root as a minimum), Boolean algebra (the ability to combine math and true/false logic in expressions; this is also known as "mixed mode" or "conditional" arithmetic), if/then or if/then/else decision-making capability, common logic operators and comparators (and, not, or, greater than, less than, equal to, greater than or equal to, less than or equal to, not equal to), and timers (the ability to utilize time delays in programs for sequencing and minimum on/off time strategies).

The "C" programming language meets all of these requirements and provides other significant features.

Features to Help the Application Programmer.

Extensive Program Documentation. The "C" language provides the capability to clearly document programs. Control sequences can be explained in detail using comments on a line-by-line basis. There is no limit to the size or number of comments used in a program. This means comments can be comprehensive enough to truly explain the program to the reader.

English-Like Vocabulary. While the textbook vocabulary of the "C" language includes items such as int,

{, !=, =, &&, and ||, "C" provides the ability to "alias" all of its commands with English words. A complete English vocabulary can be easily built by the DDC manufacturer, the controls contractor, or the end user. The result is programming that reads very English-like, for example:

IF TEMP1 ISNT SETPOINT1 THEN SET (HVAC1, TO ON);

The following table shows English aliases for some of the C commands and operators.

C Command or Operator	English Alias
>	ABOVE, AFTER
<	BELOW, BEFORE
==	IS, EQUALS
!=	ISNT (Not equal to)
&&	AND
||	OR
if	IF
}	END__IF

The Learning Curve. C, like all languages, has a specific structure and syntax that must be used for proper operation, so by itself it does not eliminate the learning curve. The advantage of C, however, is that it is taught at colleges and universities around the world. This ensures access to the necessary training and greatly expands the potential work force, because, while the selection and specification of control strategies requires a thorough understanding of HVAC systems, the development of the actual algorithm is more dependent on knowledge of the language syntax and structure. In this respect, C presents the opportunity for a single experienced HVAC engineer to lead a group of programmers in development of application programs for various installations. It also means that C goes beyond national language barriers.

Creating High-Level "Library" Functions Using C. One of the most important features of the C language is that it can be used to create a language within itself. Repetitive or commonly used tasks can be handled by creating library functions. When "called" by the program, the library function executes a predefined piece of code resulting in a specific action. As an example, consider a simple time-of-day-based on/off control algorithm. The following program entries, which are written in C using an English vocabulary, will turn an output named LIGHTS on when the time of day is between 800 and 1700 hours.

```
/* Control of LIGHTS Based on Time of Day */
IF TIME AFTER 800 AND TIME BEFORE 1700 THEN
    TURN (LIGHTS, ON);
ELSE
    TURN (LIGHTS, OFF);
END__IF
```

While this program is quite straightforward, it would be desirable to have a simple-to-use high-level function for time-based on/off control applications. For example,

TIMECONTROL(LIGHTS, 800, 1700);

Using this function the programmer specifies the name of the output to be controlled and the start and stop times. The program passes these arguments (output name, start time, and stop time) to the library function when it is executed.

Because C lets the programmer create library functions like this, the DDC manufacturer, controls contractor, or facilities engineer can, in effect, create his or her own "language" of special functions that make the programming process even easier. These functions can be changed and enhanced because they are not locked away in factory "burned" EPROM but are an actual part of the control program. This is an extremely powerful capability that can be used by the manufacturer, contractor, or end user to enhance the programmability of the system while not affecting support. C stands out among other languages in its ability to create library functions of this type. Pascal and some other languages provide a similar but much more limited library function capability. Notably, they cannot implement functions that have variable-length argument lists.

"Looping" of Functions. Another feature of C that offers significant benefits to the application programmer is the ability to "loop" multiple sets (an array) of points into programs. As an example, consider the lighting control program discussed previously.

```
/* Control of LIGHTS based on Time of Day */
IF TIME AFTER 800 AND TIME BEFORE 1700 THEN
    TURN (LIGHTS, ON);
ELSE
    TURN (LIGHTS, OFF);
END__IF
```

As written, this program controls one output named LIGHTS. With slight modification, however, this program can be used to control a number of lighting circuits. The first step is to create an array of the lighting outputs to be controlled. This is done by naming the points using the following convention: LIGHTS [1], LIGHTS [2], LIGHTS[3], LIGHTS[4], LIGHTS[5]. For this example we will assume we have five lighting circuits to control.

We will now use the "loop" instruction to execute the lighting control program for all five lighting circuits.

```
/* Control of Lighting Circuits 1-5 Using Loop Statement */
    INTEGER Liteno;
/* This defines a variable to hold the lighting circuit number */
LOOP ( Liteno, FROM 1, TO 5)
IF TIME AFTER 800 AND TIME BEFORE 1700 THEN
    TURN (LIGHTS[Liteno], ON);
ELSE
    TURN (LIGHTS[Liteno], OFF);
    END__IF
END__LOOP
```

This single program will now be executed for all five lighting circuits. This technique eliminates the need to write the program over for each lighting output.

This simple example of looping assumed that all five lighting circuits were on the same schedule. If independent start/stop times were desired, the looping function could still be used to control all five circuits. To accomplish this, it is necessary to define variables to hold the start/stop times for the five zones. These variables would use the [#] naming convention to allow them to be looped in the program. The following example demonstrates.

```
/* Control of Lighting Circuits 1-5, Independent
Schedule for each Circuit */
```

```
/* Define variables to hold start/stop times for each of the
lighting circuits */
    INTEGER Liteon [6] = {0, 800, 830, 630, 700, 700};
    INTEGER Liteoff [6] = {0, 1700, 1800, 2000, 1700,
    2200};
/* Define variable to hold lighting circuit number */
    INTEGER Liteno;
/* Loop Control Program for all Lighting Circuits */
LOOP (Liteno, FROM 1, TO 5)
    IF TIME AFTER Liteon[Liteno] AND TIME BEFORE
    Liteoff[Liteno] THEN
    TURN (LIGHTS[Liteno], ON);
    ELSE
    TURN(LIGHTS[Liteno], OFF);
    END_IF
END_LOOP
```

Off-Line Programming. Use of C as a language allows programs to be written off-line from the DDC controller. The control program is written on a workstation (personal computer) as a text file using a word processor editor. Once complete, the text file (source code) is compiled using a C compiler. The compiler checks for syntax and format errors and creates a downloadable object code file. This file is then downloaded to the controller at the appropriate time. Off-line programming and the use of a full-function word processing editor enhance the efficiency of the programming process. The programmer is not required to learn and work with a "limited" editor common to many DDC systems. In addition, the contractor does not have to have a DDC controller available for each of his programmers to work on.

A Language That is a Match for the User. The use of a programming language such as C for development of control programs in DDC systems makes even more sense when you look at the type of people who will be using it. Two groups of people interface with building control systems: system programmers and building operators. System programmers analyze the building equipment and create appropriate control algorithms. Building operators manage the day-to-day operations of the building. Their interface with the system focuses on system tuning and the change of control parameters such as start/stop times, setpoints, alarm limits, etc. Their interaction with the system does not involve the generation of control algorithms.

Because building operations personnel require so much specialized knowledge in other areas, such as HVAC equipment, maintenance and repair, and tenant environmental requirements, their job typically does not allow the time to become proficient in a programming language. This fact is made clear by the facilities management staffs as they evaluate potential systems. They look for simplicity because the people operating the system on a daily basis will not be programmers. While the case of the building operator/programmer is sometimes found, it is the exception, not the rule. Although not responsible for programming of control algorithms, some building operators may, however, want to be able to read and understand them. Here the powerful commenting features of the C language offer a significant benefit.

The systems programmer then is a technically competent person most often with some form of computer education or experience and familiar with the techniques of creating control programs. A powerful, standardized language with wide acceptance in the computer industry is the right match for this person.

Potential Disadvantages of C

While offering many advantages it is also important to note C's potential disadvantages. Use of C as a programming language for DDC systems requires the use of an intelligent workstation for the programmer, as opposed to a dumb terminal. The workstation has to have the C compiler running on it in order to create the downloadable object code version of the control program.

C is so flexible that it does not force a programming style or structure on the programmer. It is possible to write very sloppy programs in C, whereas more structured languages force certain formats and methodologies on the programmer. Good training of proper style will prevent these problems, however.

C allows the programmer to do things that, although legal to the compiler, may cause the computer to reset or "hang" or do other unexpected things when they are executed. Examples include the creation of endless loops, "weak" type checking that allows the wrong type of data to be passed to a library function, and exceeding array index limits. The programmer can create arrays of any size desired in C and could then exceed the index limit size of the array in the control program. This is a problem that would not occur until actual runtime of the program.

Off-line programming, while offering many advantages, has a drawback. Because C is a compiled language it is necessary to recompile and reload the program after making changes to the control algorithms. This forces the programmer to think out the system program more carefully to minimize time spent recompiling and reloading.

CONCLUSION

Adoption of a standard programming language for the DDC industry will offer a number of benefits to installers and end users of DDC systems. It would reduce the learning curve between systems from various manufacturers and increase the number of available programmers.

As shown in this paper, C is a strong potential candidate for use as the standard language. It is currently the most widely used programming language and provides the application programmer with an English-like vocabulary, the ability to clearly document control programs, and create functions to perform commonly used control tasks.

While other languages may exist that lend themselves to the tasks found in DDC programming they do not currently enjoy widespread acceptance, which is an important issue when choosing a standard.

REFERENCES

Kernighan, B.W., and Ritchie, D.M. 1978. *The C programming language.*

Kernighan, B.W., and Ritchie, D.M. 1988. *Byte*, Vol. 13, No. 8.

DISCUSSION

Z. Zhang, Engineer, Iowa DNR, Des Moines, IA: If C is the winner, what would be the second runner-up? Can you give a rough idea about the share of "C" in the DDC industry compared with, say, Quick Basic and others.

J.D. Petze: I would like to turn this question over to my co-author S. Chaudry.

S. Chaudry: I do not have figures on the share of the market that C has in the general computer industry. Quick Basic, while a very popular language, does not have the same features as C. One important feature it does not have is the ability to develop functions that accept variable-length arguments.

B. Chapman, ASI Controls, San Ramon, CA: Do you foresee manufacturers publishing code modules written in C to facilitate use of this language in conjunction with their hardware?

Petze: Yes I do. One of the benefits of the C language is that it will allow both manufacturers and system integrators to develop library functions (code modules) to handle new applications as they are needed. In this way the capabilities of a system can continue to evolve without changing the internal operating firmware.

I think it will also open up an aftermarket for engineering houses to publish proven control strategies and algorithms. It will also provide a new area of competition among manufacturers.