X3J16/92-0044
W621/N0121

reject; less robust w/o declaration.
mfb.

Allowing Default Arguments Anywhere

Steven E. James

April 15, 1992

As I wish to be as concise as possible, I would like to first state the essential elements of my proposal, and then expound upon my reasons later in this letter.

What I propose is a minor extension to the C++ syntax to allow a programmer to define and use parameters with default arguments that do not necessarily occur at the end of an argument list. The extension would allow a programmer to specify that default argument values should be used for a particular function call by placing commas as place holders among the arguments for a particular call. For example, you could then hypothetically specify a function call as follows:

```
myfunction1(,,arg3,,arg5)          // 1st, 2nd, & 4th arguments should
                                   // use default values
```

As you can see, this would serve two purposes... The compiler could be easily be signalled that default values for arguments are desired. Since this approach uses the same delimiter as for normal function calls, it is likely that the impact upon compiler authors in order to implement this would be relatively small. Secondly, I feel that this approach is very intuitive for the programmer and provides good readability.

Please allow me to add one additional element to this by noting that perhaps this extension could allow one to even use this proposed syntax in places where the normal "defaulting syntax" could be used, as an alternative. For example, I'd like to see the following call also be legal:

```
myfunction2(arg1,arg2,,)           // 4th and 5th arguments are explicitly
                                   // specified to use default values

myfunction3(arg1,arg2)             // Perfectly legal, but cannot tell by
                                   // looking at the function call itself
                                   // whether any arguments are to be defaulted
                                   // upon. Must review the function
                                   // prototype to make meaning of this,
                                   // assuming I can find the header file
                                   // where I put it.
```

This additional consideration would just allow the programmer another method to specify that default arguments should be used.

I have basically five reasons for proposing this:

1. Readability. This extension would improve the readability of function calls where the use of default arguments are specified. You could easily see by just looking at the function call itself which arguments are to use default values.

2. It is intuitive. By using commas as placeholders, it is obvious that the programmer has intentionally omitted certain arguments, and thus those arguments must be assuming default values.

3. It should be rather easy to implement in compilers. This extension would not require major changes in the way that compilers parse argument lists, since the compiler could just assume that whenever an empty string is found as an argument, that the programmer intends for the argument to assume a default value. Also, I cannot think of any situations where this extension would break any existing code.

4. It allows the programmer to organize the ordering of parameters in argument lists in a way that seems most natural and intuitive. Currently, if a programmer wishes to use default argument values, the ordering of the parameters in the parameter list must be set up to accommodate the use of default argument values. Thus, the use of default arguments currently imposes a restriction that my proposal would remove.

5. It allows the programmer to define a function that uses both default argument values as well as having a variable argument list, and be able to make use of both in the same function call. It is my understanding that currently this cannot be done. So, in effect, the use of default argument values in C++ currently restricts the use of variable argument lists. While I understand that default argument lists can be used now in many cases where variable argument lists were used in the past (although I have never used variable argument lists in C for the purpose of defaulting on arguments), there are still situations where I could make use of both in the same function. For example,

   ```
   int HandlePrint(int handle = Screen, int argc, ...);
   ```

   This could declare a function that receives a handle argument corresponding to different devices, and prints the contents of a variable argument list. The device handle would default to the screen device, whose handle is stored in the global const Screen.

   With my proposed syntax extension, the above hypothetical function could be called as follows:

   ```
   RetVal = HandlePrint( ThisHandle, 2, Arg1, Arg2 );
   OR
   RetVal = HandlePrint( , 2, Arg1, Arg2 );
   ```

   (many other possibilities, but above two illustrate my point)

   Please note that I am assuming that the above "HandlePrint" function should be able to deal with function calls containing an infinite number of possibilities in terms of number of arguments. I am purposely trying to show a situation where it is possible that just a mixture of function overloading and default arguments won't work.

   Also, since there are obviously more elegant ways that the printing operation performed by the above function could be done in C++, please note that the above simple function is for purposes of illustration only.

Steven E. James
4654 Coldsprings Ct., Apt. A
Columbus, OH 43220
Phone: (614) 459-2724
Compuserve ID: 70670,1025
I believe the Internet equivalent is: 70670.1025@COMPUSERVE.COM