The C Preprocessor and Jensen's Device

David W. Boyd
Department of Mathematics and Computer Science
Valdosta State College
Valdosta, Georgia 31698
DBoyd@USCN.USCN(BITNet)

Abstract

Although parameter passing by name is considered by many to be of merely historical interest since the demise of Algol 60, many of the concerns which arise with this method are also faced when programming the C preprocessor. This paper discusses some of these concerns and shows how one may simulate Jensen's Device using the C preprocessor.

Introduction

When teaching a course in Programming Languages, one generally discusses different methods for passing parameters to subprograms. Most students, being somewhat proficient in Pascal, have a basic understanding of the difference between passing parameters by value and passing them by address. Those who have programmed in C usually have acquired a better understanding of this difference through experience, since parameters are all passed by value, so that addresses must be passed explicitly when one wants to pass by address. Other parameter passing methods are new to these students. The advantages of being able to pass parameters by result are worth discussing since most students understand parameters which are used as output parameters, but not those which are restricted to be output only. It is also an interesting exercise to contrast parameter passing by address with parameter passing by value-result, using examples in which the returned values are different under these two methods. However, the method of passing parameters by name is interesting to study not only because of its place historically as the method used in Algol 60, but also because it is the basis for programming macros with replaceable parameters using the C preprocessor. Further interest in this method arises because of the difficulties in implementing and using it and the clever things which can be accomplished with it. The readability and understandability of programs which use this method can be adversely affected unless careful attention is paid to the details. The advantages of this method seem to have been outweighed by its disadvantages, resulting in the abandonment of this method after Algol 60. And yet, the concept of passing parameters by name has reappeared in the C preprocessor where textual substitution into parameterized macros is used.

Passing by Name in Algol 60

Algol 60 provided for parameter passing by name. The Algol 60 report describes the method of parameter passing by name as follows [NAUR]:

1. Any formal parameter not quoted in the value list is replaced, throughout the procedure body, by the corresponding actual parameter, after enclosing this latter in parentheses wherever syntactically possible. Possible conflicts between identifiers inserted through this process and other identifiers already present within the procedure body will be avoided by suitable systematic changes of the latter identifiers involved.

2. Finally, the procedure body, modified as above, is inserted in place of the procedure statement and executed. If the procedure is called from a place outside the scope of any nonlocal quantity of the procedure body the conflicts between the identifiers inserted through this process of body replacement and the identifiers whose declarations are valid at the place of the procedure statement or function designator will be avoided through suitable systematic changes of the latter identifiers.

The standard example found in many texts on programming languages which shows the difficulty in writing subprograms which perform correctly when using name-type parameters is the following swap procedure (written in Pascal-like syntax).

procedure Swap( name A,B:integer);
var
T: integer;
begin      {Swap}
  T := A;
  A := B;
  B := T
end;       {Swap}

Normal types of calls will perform without any problem:
  Swap(M,N);
  Swap(Q[i], Q[j]);

A call like Swap(A,T); will necessitate the renaming of
the local variable T, but will perform correctly. The
problem occurs with calls like these:
  Swap(Q[i], i);
  Swap(i, Q[i]);

The former call will be replaced with

var
  T: integer;
begin      {Swap}
  T := Q[i];
  Q[i] := i;
  i := T
end;       {Swap}

which performs as expected. However, the latter call is
replaced with

var
  T: integer;
begin      {Swap}
  T := Q[i];
  Q[i] := i;
  i := T
end;       {Swap}

where the index, i, into the array Q is (possibly) changed
prior to the assignment of the value to Q[i], giving un-
expected results. According to Sebesta, it has been
proven that is not possible to write a general procedure
using nametype parameters which will correctly inter-
change the values of its two parameters [SEBESTA].
Other disadvantages of this method are the difficulty in
implementation, and the lack of readability and
understandability of programs written using this method.

There is, however, an advantage to passing
parameters by name: flexibility for the programmer. Jen-
sen [JENSEN & NAUR] showed that one may perform
some rather clever trickery by passing parameters by
name. Examples of this type have become known as
Jensen's Device. Using a routine text as an actual
parameter in ALGOL 68 is essentially equivalent to
Jensen's Device, though a lot less sneaky [TANE-
BAUM]. The following function, written in Pascal-like
syntax exemplifies Jensen's Device.

function Sum (name Expr: real;                        name
  Term:integer;
  Increment,
  Lbound,
  Ubound: integer):real;

var
  LocalSum: real;

begin {Sum}
  LocalSum := 0.0;
  Term := Lbound;

  repeat
    LocalSum := LocalSum + Expr;
    Term := Term + Increment;
  until Term > Ubound;

  Sum := LocalSum
end; {Sum}

The following invocations show the flexibility of
parameter passing by name.

Sum(X, i, 1, 1, 10)    10*X
Sum(i*i, i, 1, 0, 10)  Σ i² from                      i = 0 to 10
Sum(i, i, 2, 1, 99)    Sum of first 50                      odd
natural numbers
Sum(A[i]*B[i], i, 1, 1, N)  dot product                      of
vectors A and B
Sum(A[i,i], i, 1, 1, N) Trace of NxN                      matrix
A.

Carrying out the textual substitution to replace each of
these calls is an instructive exercise in understanding
parameter passing by name. However, being able to write
a program and observe its behavior using name-type
parameters gives students a better understanding of this
topic. Not having an Algol 60 compiler readily available,
this was not possible, as well as impractical.

Jensen's Device

The C Preprocessor
The C preprocessor provides for macro expansion, conditional compilation and file inclusion. In accomplishing macro expansion, the preprocessor [SETHI]

1. replaces formal macro parameters textually by their corresponding actual parameters, and

2. textually substitutes the resulting body of the macro for the call.

These two steps are very similar to the procedure by which Algol 60 handles name-type parameters. However, the preprocessor makes no attempt to resolve conflicts between local variables, nonlocal variables, and actual parameters. This makes implementation easier, but still leaves the problems concerning readability and understandability. The swap macro is the canonical example once again.

```
#define SWAP(A,B)  {  int T;
    T = A;
    A = B;
    B = T;  }
```

The call

```
SWAP (i, Q[i])
```

is expanded to

```
{ int T; T = i; i = Q[i]; Q[i] = T; }
```

which gives the same erroneous results as in the Algol 60 example.

The call

```
SWAP (a, T)
```

causes even more trouble, since there is no checking for conflicts with local identifiers. In fact, the values are not swapped at all. Further problems occur when one is careless in leaving out necessary parentheses. For example,

```
#define SQUARE(X)  X*X
```

yields unexpected results when called as

```
y = SQUARE(z + 1);
```
since the preprocessor expands this call to

```
y = z + 1*z + 1;
```

(Note that Algol 60 avoids this problem by enclosing z + 1 in parentheses before substitution.) The squaring macro is correctly defined by

```
#define SQUARE(X)  (X)*(X)
```

In the above example, note that the expression z + 1 is evaluated twice. This is certainly bad if the expression involves calls to functions with side effects, or auto-incremented variables. Lexically, there can be no space between the macro name and the left parenthesis [KERNIGHAN & RITCHIE].

Because of the similarity of the problems encountered while programming the C preprocessor to those discussed in passing parameters by name in Algol 60, it seemed like a reasonable exercise to try to simulate Jensen's Device by programming using the C preprocessor. The ability not only to run programs using Jensen's Device, but also to be able to examine the output of the preprocessor makes this an excellent programming assignment and a learning experience for the students. The following is a C program with Jensen's Device encoded for the preprocessor.

```
#include <stdio.h>
#include <math.h>
#define SUM(Expr, Term,     
       Lbound, Ubound, 
       Inc, Total)     
       for(Total = 0, Term = Lbound; 
          Term <= 
          Total += (Expr) 
       
main()
{
    float X,
    a[]={1,2,3,4,5,6,7},
    b[]={1,-2,3,-4,5,-6,7};
    float T;
    SUM(X,X,1,100,2,T);
    printf("Sum odds 1 thru 99: %g\n",T);
    SUM(1.0/a[X],X,0,6,1,T);
    printf("Sum of reciprocals: %g\n",T);
    SUM(a[X]*b[X],X,0,6,1,T);
```

main()
{
    float X,
a[]={1,2,3,4,5,6,7},
b[]={1,-2,3,-4,5,-6,7};

    float T;
    for(T = 0, X = 1; X <= 100; X += 2)
        T += (X);
    printf("Sum odds 1 thru 99:\%g\n",T);

    for(T = 0, X = 0; X <= 6; X += 1)
        T += (1.0/a[X]);
    printf("Sum of reciprocals: %g\n",T);

    for(T = 0, X = 0; X <= 6; X += 1)
        T += (a[X]*b[X]);
    printf("Dot product: %g\n",T);
}

There are other things which can be accomplished using the C preprocessor. Gehani illustrates the power of the C preprocessor in defining generic functions, which can reduce programming effort, and make programs more manageable [GEHANI]. However, one must be aware of the pitfalls of programming macros with replaceable parameters.

Conclusion

The similarities between passing parameters by name in Algol 60 and programming macros with replaceable parameters using the C preprocessor have been discussed. There is no shortage of examples which can be used to illustrate the power of this method, of which Jensen's device is a prime example. Although these methods are not for beginners, and can easily lead to obfuscation, more experienced programmers can gain an appreciation for these methods by using the C preprocessor. This will give them a deeper insight into programming in general, and to programming in C in particular.

References


JENSEN, J. and NAUR, P. "Call by name: An implementation of ALGOL 60 Procedures," BIT, volume 1 (1968), 38.


