International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 3 Issue: 10 5972 - 5975

Demystifying the Sizeof () operator in C through Assembly Code

Harsh Patel
harshpatel32148@yahoo.com

AnubhavChaturvedi
acanubhav@gmail.com

Mr.VishwasRaval
vishwas.raval-cse@msubaroda.ac.in
Assistant Professor

Department of Computer Science & Engineering,
Faculty of Technology & Engineering,
The MS University of Baroda, Baroda

Abstract— Sizeof() is extensively used in C Proramming Language. However, many of the prgrammers might not be aware of the use and the power of sizeof(). The peculiar thing about sizeof() is that looks like a function and takes an argument as one of the data type, primitive or non-primitive, and operates. But the fact is that it is, actually, a compile time unary operator, which can be used to compute the size.It returns the total memory allocated for a particular object or a data type in terms of bytes. This paper throws light on the power and related matters of sizeof() and tries to decode the mystery.

Keywords- *sizeof,objdump,Cprogramming, assembly* .

*****

I. INTRODUCTION

Programmers often use Sizeof() operators in their daily programming practices, from Embedded Programming , competitive coding or web development ,sizeof is used in used in every programming language, just the syntax differs from language to language. Whether it is to find the total number of records in a database, or it is allocating memory for a linked list implementation ,from a undergraduate college student to a full time software developer everyone uses the sizeof() operator . This was used in Unix , as entire Unix was written in C but still the Unix/Linux manual pages do not contain any information about sizeof() operator ,because for a beginner it may seem to look like a function but actually it is an operator. The resultant type of sizeof() operator is size_t.Sizeof() ,can be applied for primitive data types(such as char,floats,ints) including their pointers and also for compound data types (such as structures and unions too).

II. USAGE IN C PROGRAMMING

The sizeof() operator can be used in two different cases depending upon the operand types. Let's see some examplesfor the sizeof() operator.

(Note : All codes are compiled in the gcc compiler version: gcc version 4.8.4 (Ubuntu 4.8.4-2ubuntu1~14.04)).

Case 1:When the operand is a type-name.

When sizeof() operator is used with 'type-name' as the operand (such as char,int,float,doble,etc), it returns the amount of memory that will be used by an object of that type. In this case the operand should be enclosed within parenthesis.

Code 1

```
#include<iostream>
#include<stddef.h>

using namespace std;

int main()
{
        cout<<"Sizeof(char) : "<<sizeof(char)<<endl;
        cout<<"Sizeof(int) : "<<sizeof(int)<<endl;
        cout<<"Sizeof(float) : "<<sizeof(float)<<endl;
        cout<<"Sizeof(double) : "<<sizeof(double)<<endl;
}
```

```
anubhav@linux:~/sizeof> g++ sizeof1.cpp -o size1
anubhav@linux:~/sizeof> ./size1
Sizeof(char) : 1
Sizeof(int) : 4
Sizeof(float) : 4
Sizeof(double) : 8
```

The return value of the sizeof() operator is implementation-defined ,and its type is size_t ,defined in <stddef.h>.

Case 2:When the operand is an expression.

When sizeof() is used with expression as an operand, the operand can be enclosed with or without parenthesis. The syntax of how the sizeof() is used n expression remains the same as the Case 1.

Code 2

```
#include<iostream>
#include<stddef.h>

using namespace std;

int main()
{
        int a=10;
        double d=12.34d;

        cout<<"Sizeof(a+d) : "<<sizeof(a+d)<<endl;

}
```

```
anubhav@linux:~/sizeof> g++ sizeof2.cpp -o size2
anubhav@linux:~/sizeof> ./size2
Sizeof(a+d) : 8
```

From the above code,it is very clear that when the sizeof() operator is applied to an expression , it yields a result that is the same as if it had been applied to the type-name of the resultant of the expression. Since, at compile time the compiler analyses the expression to determine its type,but it will never evaluate the expression which take place at runtime.

In the example shown in Code 2, 'a' is of int type and 'd' is of double type. When type conversion is applied as usual, the lower rank data-type is promoted to a higher rank data-type and the resultant data-type is nothing but double in our case;hencesizeof(a+d) yields double in our case ,which is 8 bytes. In general, if the operand contains operators that perform type conversions, the compiler considers these conversions in determining the type of the expression.

III. Unique Behaviour

The sizeof() operator behaves differently in comparison with other operators. The uniqueness of this operator can be illustrated by two real time programming examples. Case-1 is about the compile-time behaviour and the Case-2 is about the runtime behaviour.

Note:
In order to generate assembly code of any program we can use the '-save-temps' option,which indeed generates the "filename.s" file where filename is the name of the program file, filename.s contains only assembly code..
But in case we want to generate the assembly code parallel to the high level language code of the program we need to use 'objdump' command as mentioned in the Linux manual pages.,
+ gcc -g filename.c -o outputfile
+objedump -S outputfile
These two commands generate both in parallel.

Case 1:
Code 3

```
#include<iostream>
#include<stddef.h>

using namespace std;

int main()
{
        int a=10;

        size_t size=sizeof(a++);

        cout<<"Size of a :"<<size<<endl;
        cout<<"value of a: "<<a;

}
```

```
anubhav@linux:~/sizeof> g++ sizeof3.cpp -o size3
anubhav@linux:~/sizeof> ./size3
Size of a :4
value of a : 10
```

As variable 'a' is the alone incremented in the line so its value should get changed to 11, but out program gives output 10 , so this can be further investigated in the the assembly code.

```
anubhav@linux:~/sizeof> g++ -g sizeof3.cpp -o size3
anubhav@linux:~/sizeof>objdump -S size3


000000000040087d <main>:
#include<stddef.h>

using namespace std;

int main()
{
  40087d:       55                      push   %rbp
  40087e:       48 89 e5                mov    %rsp,%rbp
  400881:       48 83 ec 10             sub    $0x10,%rsp
int a=10;
  400885:       c7 45 fc 0a 00 00 00            movl   $0xa,-
0x4(%rbp)

size_t size=sizeof(a++);
  40088c:       48 c7 45 f0 04 00 00            movq   $0x4,-
0x10(%rbp)
  400893:       00

cout<<"Size of i :"<<size;
  400894:       be b4 09 40 00                  mov
$0x4009b4,%esi
  400899:       bf 80 10 60 00                  mov
$0x601080,%edi
  40089e:       e8 cd feffff            callq
400770<_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_
ES5_PKc@plt>
  4008a3:       48 8b 55 f0             mov    -0x10(%rbp),%rdx
  4008a7:       48 89 d6                mov    %rdx,%rsi
  4008aa:       48 89 c7                mov    %rax,%rdi
  4008ad:       e8 cefeffff             callq
400780<_ZNSolsEm@plt>
cout<<"value of a: "<<a;
  4008b2:       be c0 09 40 00                  mov
$0x4009c0,%esi
  4008b7:       bf 80 10 60 00                  mov
$0x601080,%edi
  4008bc:       e8 affeffff             callq
400770<_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_
ES5_PKc@plt>
  4008c1:       8b 55 fc                mov    -0x4(%rbp),%edx
  4008c4:       89 d6                   mov    %edx,%esi
  4008c6:       48 89 c7                mov    %rax,%rdi
  4008c9:       e8 42 feffff            callq
400710<_ZNSolsEi@plt>
}
```

The value is 10 because sizeof() operator is evaluated at the compile time and gets replace by the value '4' .Assmebly code:

```
size_t size=sizeof(i++);
40088c: 48 c7 45 f0 04 00 00 movq $0x4,-0x10(%rbp)
400893: 00
```

Finally, there are no instructions for 'a++',which is supposed to be evaluated at the runtime.

Case 2:
Code 4

```
#include<iostream>
#include<stddef.h>

using namespace std;

int main()
{
        int size;
```

```
        size_tarray_size;
        cout<<"Enter size";
        cin>>size;

        int array[size];

        array_size=sizeof(array);

        cout<<"sizeof the array is"<<array_size;


}
```

```
anubhav@linux:~/sizeof> g++ sizeof4.cpp -o size4
anubhav@linux:~/sizeof> ./size4
Enter size 6
sizeof the array is 24
```

In this the sizeof() operator is evaluated at the runtime and so we can see some really different changes in the assembly code.

```
anubhav@linux:~/sizeof> g++ -g sizeof3.cpp -o size3
anubhav@linux:~/sizeof>objdump -S size3

00000000004008bd <main>:
#include<stddef.h>

using namespace std;

int main()
{
  4008bd:       55                      push   %rbp
  4008be:       48 89 e5                mov    %rsp,%rbp
  4008c1:       41 57                   push   %r15
  4008c3:       41 56                   push   %r14
  4008c5:       41 55                   push   %r13
  4008c7:       41 54                   push   %r12
  4008c9:       53                      push   %rbx
  4008ca:       48 83 ec 28             sub    $0x28,%rsp
  4008ce:       48 89 e0                mov    %rsp,%rax
  4008d1:       48 89 c3                mov    %rax,%rbx
        int size;
        size_tarray_size;
        cout<<"Enter size";
  4008d4:       be 84 0a 40 00                  mov
$0x400a84,%esi
  4008d9:       bf a0 11 60 00                  mov
$0x6011a0,%edi
  4008de:       e8 bdfeffff             callq
4007a0<_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_
ES5_PKc@plt>
        cin>>size;
  4008e3:       48 8d 45 b4                     lea
-0x4c(%rbp),%rax
  4008e7:       48 89 c6                mov    %rax,%rsi
  4008ea:       bf 80 10 60 00                  mov
$0x601080,%edi
  4008ef:e8 cc feffff           callq  4007c0<_ZNSirsERi@plt>

        int array[size];
  4008f4:       8b 45 b4                mov    -0x4c(%rbp),%eax
  4008f7:       48 98                   cltq
  4008f9:       48 8d 48 ff             lea    -0x1(%rax),%rcx
  4008fd:       48 89 4d c8             mov    %rcx,-0x38(%rbp)
  400901:       48 89 c8                mov    %rcx,%rax
  400904:       48 83 c0 01             add    $0x1,%rax
  400908:       49 89 c6                mov    %rax,%r14
  40090b:       41 bf 00 00 00 00               mov
$0x0,%r15d
  400911:       48 89 c8                mov    %rcx,%rax
  400914:       48 83 c0 01             add    $0x1,%rax
  400918:       49 89 c4                mov    %rax,%r12
  40091b:       41 bd 00 00 00 00               mov
$0x0,%r13d
  400921:       48 89 c8                mov    %rcx,%rax
  400924:       48 83 c0 01             add    $0x1,%rax
  400928:       48 c1 e0 02             shl    $0x2,%rax
  40092c:       48 8d 50 03                     lea
0x3(%rax),%rdx
  400930:       b8 10 00 00 00                  mov
$0x10,%eax
  400935:       48 83 e8 01             sub    $0x1,%rax
  400939:       48 01 d0                add    %rdx,%rax
  40093c:       be 10 00 00 00                  mov
$0x10,%esi
  400941:       ba 00 00 00 00                  mov    $0x0,%edx
  400946:       48 f7 f6                div    %rsi
  400949:       48 6b c0 10             imul   $0x10,%rax,%rax
  40094d:       48 29 c4                sub    %rax,%rsp
  400950:       48 89 e0                mov    %rsp,%rax
  400953:       48 83 c0 03             add    $0x3,%rax
  400957:       48 c1 e8 02             shr    $0x2,%rax
  40095b:       48 c1 e0 02             shl    $0x2,%rax
  40095f:       48 89 45 c0             mov    %rax,-0x40(%rbp)
        array_size=sizeof(array);

        cout<<"sizeof the array is"<<array_size;


}
  400963:       48 89 c8                mov    %rcx,%rax
  400966:       48 83 c0 01             add    $0x1,%rax
        cout<<"Enter size";
        cin>>size;

        int array[size];

        array_size=sizeof(array);
  40096a:       48 c1 e0 02             shl    $0x2,%rax
  40096e:       48 89 45 b8                     mov        %rax,-
0x48(%rbp)

        cout<<"sizeof the array is"<<array_size;
  400972:       be 8f 0a 40 00                  mov
$0x400a8f,%esi
  400977:       bf a0 11 60 00                  mov
$0x6011a0,%edi
  40097c:       e8 1f feffff            callq
4007a0<_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_
ES5_PKc@plt>
  400981:       48 8b 55 b8                     mov                -
0x48(%rbp),%rdx
  400985:       48 89 d6                mov    %rdx,%rsi
  400988:       48 89 c7                mov    %rax,%rdi
  40098b:       e8 20 feffff            callq
4007b0<_ZNSolsEm@plt>
  400990:       48 89 dc                mov    %rbx,%rsp
}
```

The change in the code when sizeof() runs at the runtime is

```
        array_size=sizeof(array);
  40096a:       48 c1 e0 02             shl    $0x2,%rax
  40096e:       48 89 45 b8                     mov        %rax,-
0x48(%rbp)
```

IV. THENEEDOFsizeof()

Case 1:
Auto determination of the number of elements in an array.

```
#include<iostream>
#include<stddef.h>

using namespace std;

int main()
{
int array[]={10,,20,30,40|;

cout<<"number of elements in array are "<<sizeof(array)/sizeof(array[0]);

}
anubhav@linux:~/sizeof> g++ sizeof5.cpp -o size5
anubhav@linux:~/sizeof> ./size4
number of elements in array are 4
```

Hence Sizeof() can be directly used to find the size of the array.

Case 2:
To allocate a block of memory dynamically of a particular data type.

For dynamic memory allocation in the case of arrays , if we want t to allocate a block of memory that is big enough to hold the integers in an array, sizeof comes in handy and is of great help , since we do not know the exact sizeof(int) due to architectural difference in 64-bit or 32 bit implementation. We can dynamically allocate memory using malloc function for a particular archietecture.

```
Int * ptr =malloc(5*sizeof (int));
```

Case 3:

Sometimes it is very difficult to predict the sizes of compound data types such as structure, due to structure padding and to predict the size of the unions. Sizeof() is of great use here. Structures are often used for linked list implementations and unions for programming in the embedded systems.

Cases when sizeof() will not work:

1. Case1:If one tries to find the size of the bit field, it would lead to a segmentation with core dumped.
2. Case 2:If one tries to find the size of a function, than sizeof again will lead to error.
3. Case 3:If one tries to find the size of an incomplete type(void), than sizeof again will lead to error.

How sizeof() is different from a function call ?

Let us consider the following code to understand how sizeof() is different from a function call.

```
#include<iostream>
#include<stddef.h>

using namespace std;

int main()
{
        int x=2;
        cout<<"Sizeof x is : "<<sizeof x<<endl;
        cout<<"Sizeof 5 is : "<<sizeof 5<<endl;

}

anubhav@linux:~/sizeof> g++ sizeof1.cpp -o size1
anubhav@linux:~/sizeof> ./size1
Sizeof x is : 4
Sizeof 5 is : 4
```

In the above example, sizeof() will work even if the braces are not present across the operands ,whereas in functions the braces are the must. So size of is not a function because:

(i)It can be applied for any type of operand.
(ii)It can also be used when type is an operand.
(iii)No brackets needed across operands.

Sizeof() operator can also be implemented as a macro which leads to a faster execution as compared to a function.

```
#include<iostream>
#include<stddef.h>

using namespace std;

#define my_sizeof(type) (char *)(&type+1)-(char*)(&type)
int main()
{
double x;
printf("%d", my_sizeof(x));
getchar();
return 0;
}
```

V. CONCLUSION

The sizeof() operator gives the size in bytes of its operand given as an argument. This argument could be an expression or the parenthesised name of a type. If the type of the operand is a variable length array type or a vector, the operand is evaluated at runtime and returns the result accordingly.

REFERENCES

[1] **http://www.geeksforgeeks.org/**
[2] **The 8051 Microcontroller (Merrill's international series in engineering technology)**
[3] **Linux Man Pages.**
[4] http://en.cppreference.com/w/c/language/sizeof