of base. By disassembly running code in VC6.0, it could be found out that local variables are accessed to throughout ebp. It should be standardized by x86 system and set by Intel.

By a simple function call, the code below is an example of how running stack works and what passing by value is:

```
int swap(int x,int y){
  int t;
  t = x;
  x = y;
  y = t;
       return 1;
}
void main(){
  int a,b,c;
  a = 1234;
  b = 2345;
  c = swap(a,b);
}
```

Firstly, let us take a look of source code in main function and its disassembling code.

```
18:          int a,b,c;
19:          a = 1234; // when running here, ebp=124500. it is main function's base also.
004012B8     mov     dword ptr [ebp-4],4D2h //to set a value to variable a( its address is 1244996)
20:          b = 2345;
004012BF     mov     dword ptr [ebp-8],929h //to set a value to variable b(address is 1244992)
21:          c = swap(a,b);
004012C6     mov          eax,dword ptr [ebp-8]
004012C9     push         eax       //push argument b(address is 1244908)
004012CA     mov          ecx,dword ptr [ebp-4]
004012CD     push         ecx       //push argument a(address is 1244904)
004012CE     call         @ILT+0(swap) (00401005) //call swap function
004012D3     add          esp,8
004012D6     mov          dword ptr [ebp-0Ch],eax //returned value stored in c
```

When running in called function:

```
9:       int swap(int x,int y){
00401250     push         ebp      //store calling function's ebp
00401251     mov          ebp,esp //set stack top to current function's base, ebp=1244896
……          //some irrelevant disassembling code is omitted.
10:          int t;
11:          t = x;
00401268     mov          eax,dword ptr [ebp+8] //x's address is ebp+8, that is 1244904
0040126B     mov          dword ptr [ebp-4],eax
12:          x = y;
0040126E     mov          ecx,dword ptr [ebp+0Ch] //y's address is 1244908
00401271     mov          dword ptr [ebp+8],ecx
13:          y = t;
00401274     mov          edx,dword ptr [ebp-4]
00401277     mov          dword ptr [ebp+0Ch],edx
14:          return 1;
0040127A     mov          eax,1      //store returned value 1 in register eax before return
15:      }
```

Throughout comparing source code of main function and swap function to their disassembling code, it could be found out how an argument is transferred to a parameter when a function call

happens. When function swap(a,b) in main function is called, both values of argument a and b are copied to running stack. However, the allocation that the copies occupied are the very same allocation that parameter x and y have. So, push operation takes a part of transferor here.

Before the called function finished its work, it stores a return value into register eax. This situation always happens whenever a return type is not void. The calling function gets return value by reading register eax as soon as the called function returned.

When an object is passed from calling function to called function, its situation is the same as a basic value is passed. Running stack is also needed. The only difference is that it needs call copy constructor at the same time. In the running stack between calling function and called function, there is an area that could be accessed to by both functions. Calling function write in as arguments before a function call happened and called function read out as parameters after the same function call. So, the only thing and necessary thing is to create passing-by-value objects in the correspondent memory segment of running stack. Objects could be transferred from calling function to called function automatically. The essence of above procedure is to create a brand new object in running stack by calling a copy constructor [5].

Teaching Method

In C++ programming, function call appears frequently. Data exchange between functions is unavoidable. It tangles learners almost all the time in their studying. All learners want to make it clear. But it is always a perplexing issue. The above example's logic is simple and clear. Even most beginners could understand it. And people with little C++ skills could implement it easily. Its correctness could be tested on any personal computers. In the process of teaching, students could have a direct recognition on function's functions, usages and its data transferring, only if they are introduced to the way function call works from the perspective of assembly and the way that hardware works.

Actually, there are still other questions such as why return value is stored in register eax instead of ebx or ecx or other registers. The author here thinks that it is just a problem of habit. Just think about that function's passing by value. Why does it use running stack other than registers? By the way, the sequence of passing by value is from back to front. It is need to know that all these possibilities could work well if the very first designer wants to do so when he made his complier. But to beginners, it could be perplexing with for a long time. Only if they understood that theory is concluded from practice and is used to instruct practice, they could go farther in computer science.

Acknowledgement

This article is sponsored by Guangdong University Characteristic Specialty Construction Foundation. Thanks for its supporting for projects of Software Engineering.

References

[1] Li Zhen, Yuan Dong, and Jiangzhou He, "C++ programming language (the 4th edition)," Tsinghua university publishing house, pp. 529–551, July 2010.

[2] Xiaohon Su, Zhigang Sun, and Huipeng Chen, "C programming language tutorial for universities (the 3rd edition)," Publishing House of Electronics Industry, pp.68–83, May 2012.

[3] Richard Johnsonbaugh and martin Kalin, "C++ Object Oriented Programming (the 2nd edition)," Tsinghua university publishing house, pp. 271–350, August 2005.

[4] D. S. Malik, "C++ Programming Program Design Including Data Structures," Publishing House of Electronics Industry, pp. 710–741, June 2003.

[5] Jiao Geng, Jing Lee, "Comparision of Passing By Value between C/ C++ and JAVA", South North Bridge, pp.151-152, August 2010.