دانشکده‌ی مهندسی کامپیوتر

# طراحی کامپایلرها

نیم‌سال دوم ۹۹ـ۹۸

نام و نام خانوادگی: حسن ذاکر، علیرضا دقیق، سپهر فعلی

پاسخ تمرین سری چهارم | **محیط‌های زمان اجرا** | موعد تحویل: ۹۹/۱۰/۱۱

## مسئله‌ی ۱.

پاسخ.

فرض کنید تابع f داریم که می‌خواهیم تعداد نامشخصی پارامتری می‌گیرد. حال ما این تابع را صدا می‌زنیم.
f(1,2,3)
نحوه فراخوانی در tac به این صورت است:

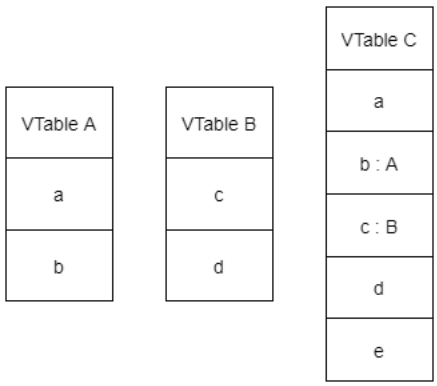
```
1 push 3
2 push 2
3 push 1
4 call f
```

حال ما این سه مقدار را روی استک داریم.در نتیجه تابع به آن دسترسی دارد. حال می‌توانیم درتابع f مکانیزمی طراحی کنیم که بتواند از روی استکبه ترتیب بخواند

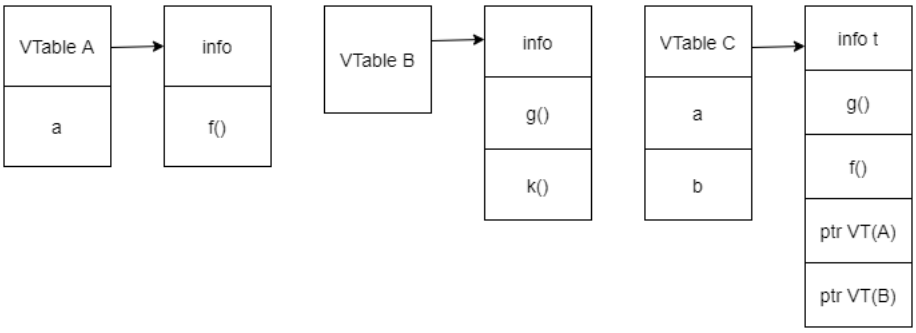
```
1 format_string <- stack[0]
2 offset <- 1
3 while (parsing):
4     token = tokenize_one_more(format_string)
5     if (needs_integer (token)):
6         value <- stack[offset]
7         offset = offset + 1
```

## مسئله‌ی ۲.

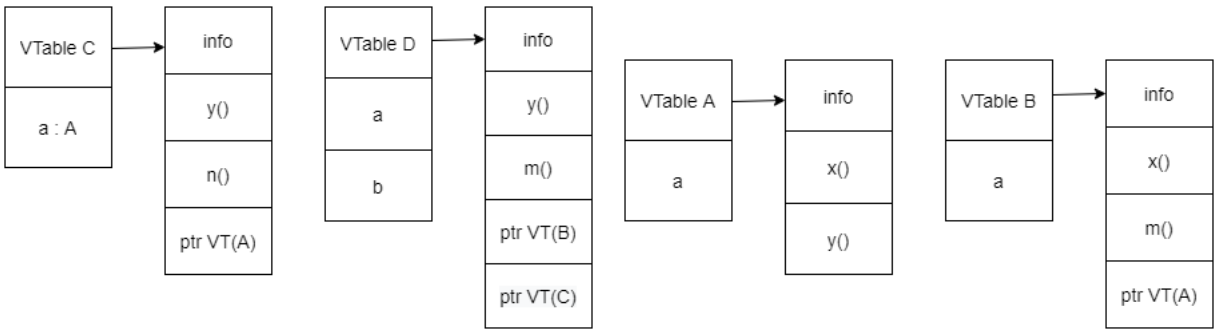
پاسخ. ابتدای هر vtable object layout آن قرار دارد که یک نشانگر به آن است. سپس در بایت‌های بعدی فیلدهای مربوط به کلاس‌های پدر به ترتیب می‌آید.(فیلدهایی که تا الان نیامدند)و در انتها هم فیلدهای مختص به کلاس مربوطه می‌آید که در بقیه فیلدها نباشد. در vtable کلاس مربوطه ابتدا اطلاعات کلاس می‌آید و سپس برای هر تبع یک pointer به کد آن را نگه میداریم. و در ادامه برای هر کلاس پدر یک pointer به کلاس پدر آن نگه می‌داریم.
هنگام استفاده از یک آبجکت از کلاس همان آبجکت مشکلی وجود ندارد. برای دسترسی به فیلدها مانند وراثت یگانه عمل می‌کنیم و هنگام فراخوانی توابع آن کلاس اگر تابع مورد نظر در آن کلاس بود که فراخوانی می‌شود اگر نبود در vtable پدران کلاس جستجو می‌کنیم. اگر بود که کال می‌شود و اگر در هیچ کلاس پدری وجود نداشت خطای مورد نظر ارسال ایجاد می‌شود.




.۱




.۲




.۳

## مسئله‌ی ۳. مساله ۳

پاسخ.
کد tac

```
1 main:
2   BeginFunc 8;  //allocate 8 bytes for locals and temporary
      registers for function main
```

```
3     _t0 = (value for parameter x); //save value of x in t0
4     PushParam _t0; //push t0 to stack frame
5     PushParam c; //push an object of C to stack frame
6     LCall _C.f; // Call function f()
7     PopParams 8; // pop parameters that we've been pushed in stack (t0
          and c)
8     EndFunc;
```

مسئله‌ی ۴.

پاسخ.

- call-by-result:

```
1  int n;
2
3  void f(int k){
4      n = n + 1
5      k = k + 4;
6      printf("n=%d, k=%d", n, k);
7      return;
8  }
9  int main(){
10     n = 0;
11     f(n);
12     print("n=%d", n);
13 }
```

خروجی:

| result by call | reference by call | |
|---|---|---|
| n=1, k=4 | n=5, k=5 | output |
| n=4 | n=5 | |

در روش call by result در واقع مقدار evaluate کردن پارامتر تابع مانند روش call by value عمل می‌کنیم و هنگام کال کردن تابع یک کپی از متغیر را پاس می‌دهیم و دستورات در تابع انجام می‌شوند بدون اینکه تغییری در مقدار متغیر اصلی ایجاد شود. سپس هنگام پایان تابع و برگشت به تابع caller مقدار نهایی آن کپی را در متغیر اصلی کپی می‌کنیم.

کد tac

```
1  _f:
2    BeginFunc 8;
3    _t0 = 1;
4    _t1 = 4;
5    n = n + _t0
```

```
6     k = k + _t1
7       PushParam n;
8     PushParam k;
9     LCall _printf;
10    PopParam 8;
11    n = k;
12    EndFunc;
13
14  main:
15    BeginFunc 4;
16    n = 0;
17    _t0 = n;
18    PushParam _t0;
19    LCall _f;
20    PopParam 4;
21    PushParam n;
22    LCall _printf;
23    PopParam 4;
24    EndFunc;
```

- call-by-name: در این روش پارامترهای تابع هنگام کال شدن تابع evaluate نمی‌شوند بلکه هر زمانی که از آن‌ها در تابع استفاده شود مقدار آنها evalute می‌شود. مزیت این روش این است که اگر اگر تابع پارمتری داشت که در تابع استفاده نشده‌باشد هرگز evalute نمی‌شود.(ممکن است یه پارامتر در واقع یک expr باشد که نیاز به محاسبه داشته باشد.)
  عیب این روش این است که اگر از یک پارامتر چندین بار در بدنه تابع استفاده شود هر بار باید آن را evalute کنیم و این باعث می‌شود از نظر زمانی بصرفه نباشد.(مخصوصا که پارامتر یک expr محاسباتی باشد آن وقت هر بار باید محاسبه شود.)

- call-by-need: مانند روش call by name است. در واقع حالت memoized شده‌ی روش بالا است و به نوعی عیب روش بالا با استفاده از thunk حل می‌کند. به این صورت که هر پارامتر هنگام اولین استفاده در بدنه‌ی تابع evaluate می‌شود و بعد از آن این مقدار ذخیره می‌شود و در دفعات بعدی استفاده از پارامتر از این مقدار استفاده می‌شود.(مقدار هر پارامتر حداگثر یکبار evalute می‌شود.)

- call-by-name Vs call-by-need

```
1  int n;
2
3  void f(int m){
4      int k = m;
5      printf("k=%d", k);
6      n = n + 1;
7      k = k + m;
8      printf("k=%d", k);
9      return;
10 }
11 int main(){
12     n = 1;
13     f(n*n);
```

```
}
```

خروجی:

| need by call | name by call | |
|---|---|---|
| k=1 | k=1 | output |
| k=2 | k=5 | |

## مسئله‌ی ۵.

پاسخ. در مواردی که طول استک متغیر است و در زمان اجرا مشخص می‌شود، نمی‌شود از یک مقدار offset ثابت از stack pointer برای دسترسی به متغیر استفاده کرد. از طرفی هنگام پاک کردن این مقادیر از استک نیاز به fp داریک تا بدانیم تا کجا را باید پاک کنیم زیرا طول متغیر است، از انتها تا fp پیس می‌رویم و حذف می‌کنیم. مثال:

```
int main(){
    int *arr = malloc(10 * sizeof(int));
}
```

TAC:

```
main:
  BeginFunc 12;
  _t0 = 10;
  _t1 = sizeof(int);
  _t2 = _t0 * _t1;
  PushParam _t2;
  LCall malloc;
  PopParams 4;
  EndFunc;
```

## مسئله‌ی ۶.

پاسخ.

```
cgen(do stmt while(expr))={
    let L_before be a new label
    let L_after be a new label
    Emit(L_before:)
    cgen(stmt)
    let t = cgen(expr)
    Emit(Ifz t GOTO L_after)
    Emit(GOTO L_before)
    Emit(L_after)
}
```