

طراحی کامپایلرها

نیمسال دوم ۹۸-۹۹

نام و نام خانوادگی: حسن ذاکر



دانشکده مهندسی کامپیوتر

موعده تحویل: ۹۹/۱۰/۱۱

محیطهای زمان اجرا

پاسخ تمرین سری چهارم

مسئله ۱. طراحی DFA

پاسخ.

مسئله ۲.

پاسخ.

۱.

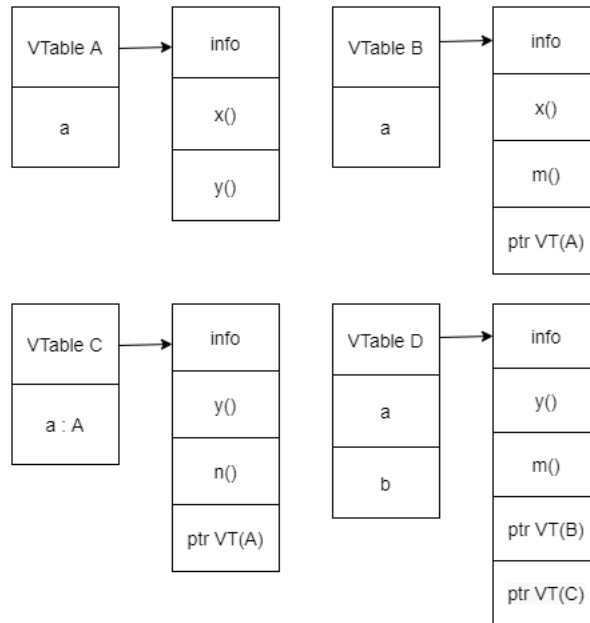
۲.

۳.

مسئله 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);
4   PushParam _t0;
5   PushParam c;
6   LCall _C.f;
7   PopParams 8;
8   EndFunc;

```

مسئله 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 پارامتری داشت که در تابع استفاده نشده باشد هرگز **evalute** نمی‌شود. (ممکن است به 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 بصرغه نباشد. (مخصوصاً 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 حداکثر 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);
14 }

```

خروجی:

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

مسئله‌ی ۵.

پاسخ.

مسئله‌ی ۶.

پاسخ.

```

1  cgen(do stmt while(expr))={
2      let L_before be a new label
3      let L_after be a new label
4      Emit(L_before:)
5      cgen(stmt)
6      let t = cgen(expr)
7      Emit(Ifz t GOTO L_after)
8      Emit(GOTO L_before)
9      Emit(L_after)
10 }

```