دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

موضوع

تعریف پروژه‌ی درس اصول طراحی کامپایلر (توابع)

نگارش

امیرمحمد نظری

نیما داوری

استاد

دکتر محمدرضا رزازی

پاییز ۱۳۹۸

در نوشته‌ی پیش رو به نحوه‌ی پیاده‌سازی توابع برای پروژه‌ی درس اصول طراحی کامپایلر می‌پردازیم. لازم به ذکر است که ساختمان داده‌ی پشته بهترین ساختار داده برای انجام عملیات‌های مرتبط با توابع می‌باشد. می‌توان از پشته‌های متفاوتی برای سهولت در پیاده‌سازی توابع استفاده کرد. نحوه‌ی پیاده‌سازی توابع را در سه مرحله توضیح می‌دهیم.

در مرحله‌ی اول به عملیات‌های مورد نیاز هنگام فراخوانی یک تابع می‌پردازیم. ابتدا باید متغیرهای موجود در برنامه در داخل پشته ذخیره شوند. سپس آدرس بازگشت در پشته ذخیره می‌گردد. در قدم بعدی ورودی‌های تابع در پشته ذخیره می‌گردند. در نهایت با یک پرش به سمت بدنه اصلی تابع می‌رویم.

در مرحله‌ی دوم به عملیات‌های مورد نیاز در بدنه‌ی اصلی تابع می‌پردازیم. ابتدا ورودی‌های تابع را برای عملکرد صحیح تابع از پشته بازیابی می‌کنیم. سپس بدنه‌ی اصلی تابع را اجرا می‌نماییم. در ادامه، آدرس بازگشت را از پشته خارج کرده و پس از آن خروجی تابع را در داخل پشته ذخیره می‌کنیم. حال با یک پرش به سمت آدرس بازگشت می‌رویم.

در مرحله‌ی سوم به عملیات‌های مورد نیاز پس از اجرای بدنه‌ی اصلی تابع می‌پردازیم. ابتدا خروجی تابع را از پشته بازیابی می‌نماییم. سپس مقادیر اولیه‌ی متغیرهای موجود در برنامه را از پشته بازیابی می‌نماییم. حال عملیات مربوط به تابع به پایان رسیده است.

بنابراین سه مرحله‌ی ذکر شده برای پیاده‌سازی توابع دارای اهمیت بسیار می‌باشد. می‌توانید از قطعه کدهای زیر برای پیاده‌سازی مرحله‌های ذکر شده، الگو بگیرید.

Push

```
top = top – 1
*top = T0
```

Pop

```
T0 = *top
top = top + 1
```

Return

```
// Pop return address.
returnAddress = *top
top = top + 1
// Push return value.
top = top – 1
*top = returnValue
goto returnAddress
```

Function/Procedure call

```
// Store program state.
top = top – 1
*top = var1
top = top – 2
*top = var2
…
// Push return address.
top = top – 1
*top = newLabel
// Push parameters.
top = top – 1
*top = param1
top = top – 2
*top = param2
…
// Jump to function/procedure.
goto label
// Pop return value when returned.
newLabel : T0 = *top
top = top + 1
```

زبان C قابلیت اشاره به Label ها را ندارد اما GCC این قابلیت را برای آن به شکل زیر ایجاد کرده است.

```
void* returnAddress = &&L0;
```

```
goto *returnAddress; // Jumps to L0.
```

برای پیاده‌سازی این بخش می‌توانید استک مربوط به Label ها را از داده‌ها جدا در نظر بگیرید. این استک آرایه‌ای از *void ها خواهد بود.

حال برای شفافیت بیشتر از یک مثال استفاده می‌کنیم. قطعه برنامه فاکتوریل زیر را در نظر بگیرید. (فایل‌های ورودی و خروجی در همین پوشه قرار داده شده‌اند.)

```
class FunctionCall{
	int num = 5;

	int factorial(int n){
		if(n < 2)
			return 1;
		int value = factorial(n - 1);
		return n * value;
	}

	void _main(){
		int results = factorial(num + 2);
		print("{results}");
	}
}
```

خروجی مورد نظر می‌تواند برنامه‌ی زیر باشد.

```
#include <stdio.h>
#include <stdlib.h>

int main(){
double num, n, value, results, T0, T1, T2, T3, T4;
double* stackTop = malloc(10000);
void** labelStackTop = malloc(10000);
void* returnAddress;
num = 2;
goto _main;
factorial:
// Pop parameters.
n = *stackTop;
stackTop = stackTop + 1;

if(n < 2) goto L1;
goto L2;
// Pop return address.
L1: returnAddress = *labelStackTop;
labelStackTop = labelStackTop + 1;
// Push return value.
stackTop = stackTop - 1;
*stackTop = 1;
```

```
// Return.
goto *returnAddress;
// Store program state;
L2: stackTop = stackTop - 1;
*stackTop = n;
// Store return address.
labelStackTop = labelStackTop - 1;
*labelStackTop = &&L3;
// Push parameters.
T2 = n - 1;
stackTop = stackTop - 1;
*stackTop = T2;
// Call function.
goto factorial;
// Return from factorial and pop the returned value.
L3: T3 = *stackTop;
stackTop = stackTop + 1;
value = T3;
// Restore program state.
n = *stackTop;
stackTop = stackTop + 1;
T4 = n * value;
// Pop return address.
```

```
returnAddress = *labelStackTop;
labelStackTop = labelStackTop + 1;
// Push return value.
stackTop = stackTop - 1;
*stackTop = T4;
// Return.
goto *returnAddress;
_main:
// Store program state.
/* Empty */
// Store return address.
labelStackTop = labelStackTop - 1;
*labelStackTop = &&L0;
// Push parameters.
T0 = num + 2;
stackTop = stackTop - 1;
*stackTop = T0;
// Call function.
goto factorial;
// Return from factorial and pop the returned value.
L0: T1 = *stackTop;
stackTop = stackTop + 1;
results = T1;
```

```
printf("%lf\n", results);
// Terminate the program.
goto end;
end: return 0;
}
```

پایان