Procedures

CS152

Chris Pollett

Nov. 26, 2008.

Outline

- Procedure and Functions
- Activations
- Parameter Passing

Introduction

- Last day, we talked about basic control in programming languages.
- We looked at an distinguished between expressions and statements; and looked at different kinds of each.
- Today, we are going to look at the next level of structured control abstractions.
- We are going to look at procedures or functions.
- We are going to look at **activation records** which are the collections of data needed to maintain a single execution of a procedure at run-time

Procedures and Functions

- A procedure is a mechanism in a programming language for abstracting a group of actions.
- This group of actions is called the **body** of the procedure.
- In addition to the body, a procedure has a **specification** or interface. This lists its name, and the name and type of its parameters, and the return type if any.
- A procedure is **called** or **activated** by stating its name together with **arguments** to the call**.**
- A call to a procedure transfers control to the beginning of the body of the called procedure (**the callee**).
- When execution reaches the end of the callee, control is returned to the **caller**.
- Sometimes a distinction is made between procedures, which carry out their operations by changing their parameters or nonlocal variables; and functions, which can appear in expressions and compute returned values.

Procedure Semantics

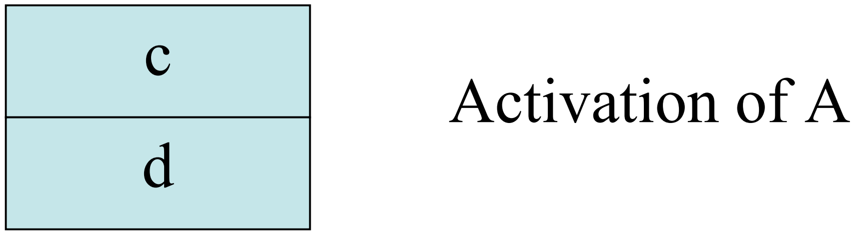
- A procedure is a block whose declaration is separated from its execution.
- So a labeled C block would not count as a procedure. A:{ int c,d;/*code*/ B:{ int d; /*morecode*/}}
- When we were talking about basic semantics, we said that the **environment** determines the allocation of memory and maintains the meanings of names during execution.
- In a block-structured language, when a block is encountered during execution, it causes the allocation of local variables and other objects corresponding to the declarations of the block.
- This allocated memory is called the **activation record** of the block and the block is said to be activated.

Example

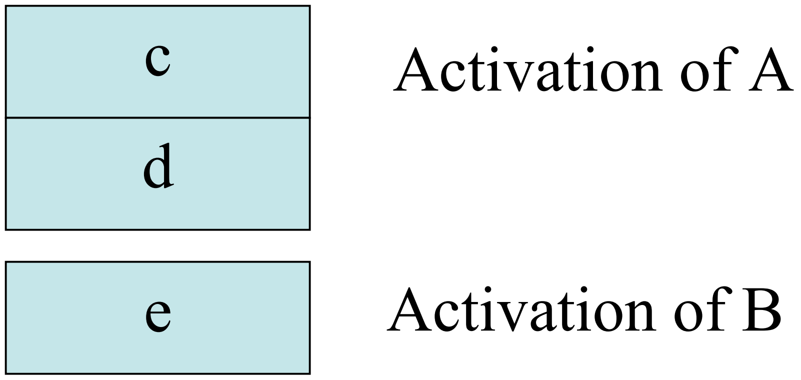
- Consider:

```
A:{ int c,d;/*code*/ B:{ int e; /*morecode*/}}
```

- Before B begins to execute the activations in the environment might look like:



- When B begins to execute the environment would look like:



- After B finishes the environment would go back to the first situation.

Environments

- It is conpletely possible to refer to global variables in C within a block…
- This would be called a **nonlocal reference**.
- So in addition to the picture of the last slide on could imagine a global environment whose data is before each of the local activations.
- If we have functions like:

```
int d=10;
void A() {/*code*/ B();}
void B() {/*more code uses d*/}
```

- We see d is part of the environment in which B must run. This environment is called the **defining environment** of B.
- On the other hand the environment in A (also has access to d) in which B was called is referred to the **calling environment**.
- We want our runtime system to be able to keep track of these two different environments so that we can give procedures the semantics we have described a couple of slides back.

Parameters and Arguments

- There are two methods of communicating between calling blocks and defining blocks of a procedure: we can either use nonlocal references or we can use parameters.
- We have previously distinguished between **parameters** (slots to feed data into in a procedure) and **arguments** (the actual data fed into those slots).
- Sometimes parameters are called **formal parameters** and arguments are called **actual parameters**.
- Procedures which depend only on paramters and fixed language features are said to be in **closed form**.

Parameter Passing Mechanisms

- We now look at different ways data can be fed into parameters.
  - Pass by Value - replace all the parameters in the body of the procedure by the corresponding argument value
  - Pass By Reference - pass the location of the argument
  - Pass By Value Result - do pass by value, then take the final result of the procedure and copy it back to the location of the passed argument.
  - Pass By Name and Delayed Evaluation -- this is what we talked about in ML using the delay mechanism.