Implementing C-style Procedures

Declarations, Procedures, & Code Generation

Dr. William Harrison
harrisonwl@missouri.edu
CS 4430 Compilers I

Translating Declarations

- Note that Micro+ has two types occurring in declarations: int and int[]

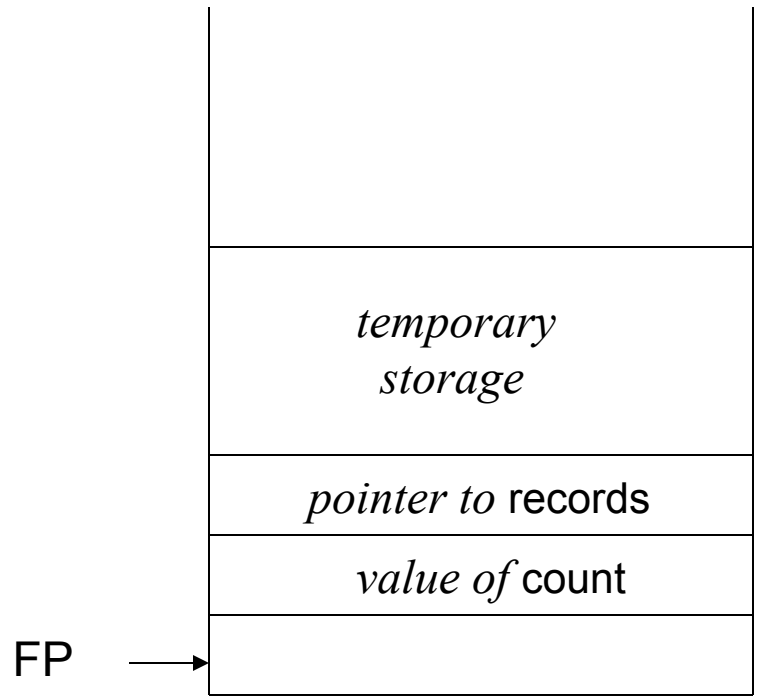
```
int count ;
int[] records[100] ;
count := 1 ;
record[count] := 5 ;
    …<rest of the program>…
```

- Given a Micro+ program, we have to decide on how that variables are represented in memory
  - That is, where & how each variable is stored and accessed

Example

In these slides, “FP” is like “EBP” and “SP” is like “ESP”

```
int count ;
int[] records[100] ;
count := 1 ;
record[count] := 5 ;
    …<rest of the program>…
```



assuming
- 1 word integers here
- a “frame pointer” register FP

C-style procedures

A high-level view of a C program (i.e., ignoring separate files) is:

```
global-declarations;
int foo (char v) { … body-foo …  }
void bar (int a) { … body-bar … }
…
void main (int argv[]) {… body-main … }
```

** ignoring C's procedure pointers by which one may hack downward/upward fun-args.*

Variable scoping in C

Scoping in C is particularly simple:

global-declarations;
int foo (char v) { *decls-foo* … ***variable*** … }
void bar (int a) { … *body-bar* … }
…
void main (int argv[]) {… *body-main* … }

Question: if a variable is used in *body-foo*, where is it declared?

Variable scoping in C

global-declarations;
int foo (char v) { *decls-foo* … ***variable*** … }
void bar (int a) { … *body-bar* … }
…
void main (int argv[]) {… *body-main* … }

Question: if a variable is used in *body-foo*, where is it declared?

procedure scoping

Procedure names are variables as well

```
global-declarations;
int foo (char v) { … proc-call …  }
void bar (int a) { … body-bar … }
…
void main (int argv[]) {… body-main … }
```

proc-call must refer to:

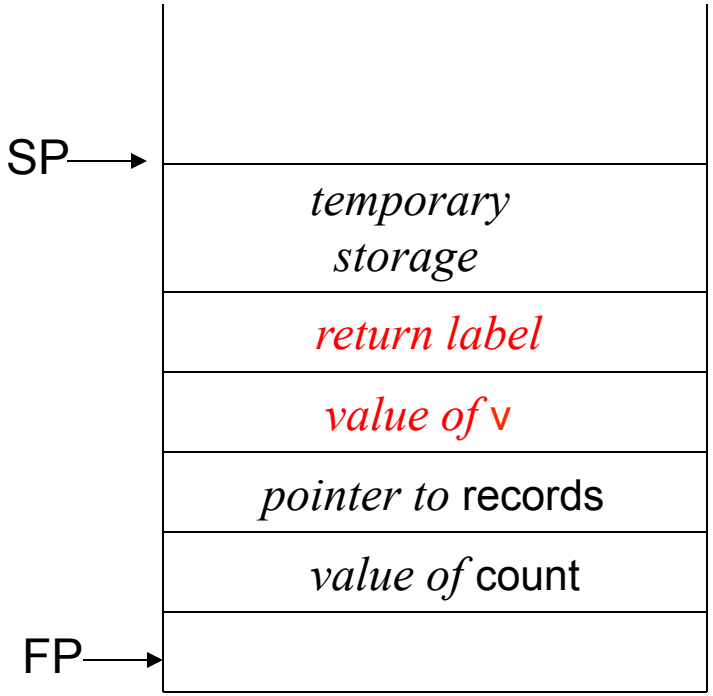
- user defined procedures (foo, …), or
- (global) library routines

Implications of C language design for compiler writer

- Compiling procedures confronts two issues
  - compiling procedure **declarations**
    - similar to what we’ve done for Micro+
    - this is deliberate – Micro+ programs look like the bodies of C procedures
    - code must be stored at a new label.
  - compiling procedure **calls**
    - create “activation record” for call
    - keep track of return label
    - jump to the code
- The simple scoping of C-procedures simplifies their compilation
  - main issue: keeping track of variables during execution

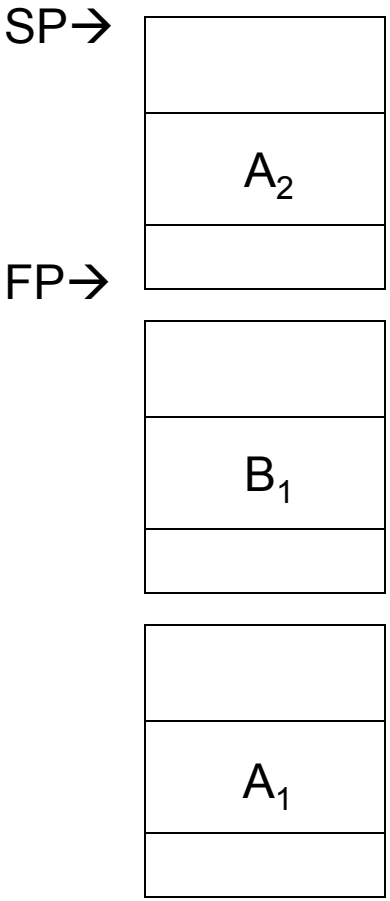
Activation record

```
void foo (char v) {
    int count ;
    int[] records[100] ;
    count := 1 ;
    record[count] := 5 ;
     …<rest of foo>…
}
```



Execution generates a new activation record for **each** call

```
proc A (aargs) {…B(bv)…}
proc B (bargs) {…A(av)…}
begin
    A (…); // a procedure call
end
```



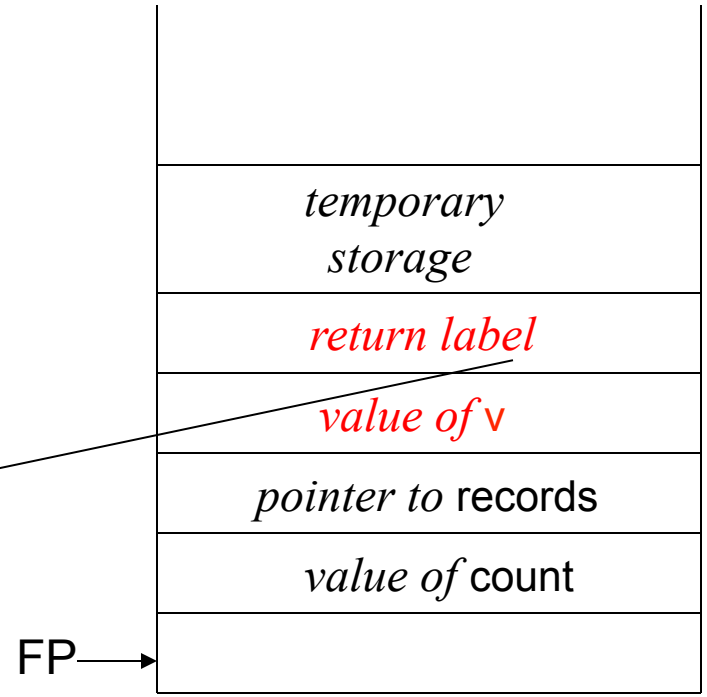
Keep track of the "current activation"

- with frame pointer FP
- stack pointer SP
- calling code must prepare this new AR

Activation record

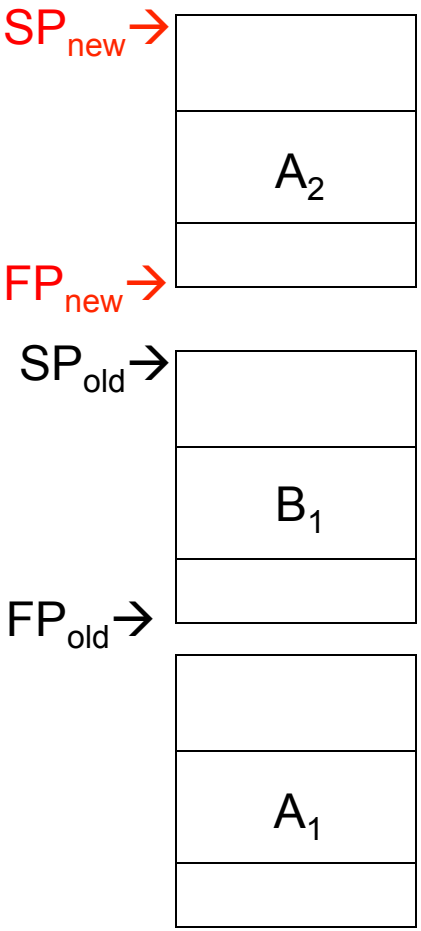
```
void foo (char v) {
    int count ;
    int[] records[100] ;
    count := 1 ;
    record[count] := 5 ;
     …<rest of foo>…
}
```

This is label from the caller's code



making a call: when B calls A

```
proc A (aargs) {…B(bv)…}
proc B (bargs) {…A(av)…}
begin
   A (…);
end
```



Code for call must

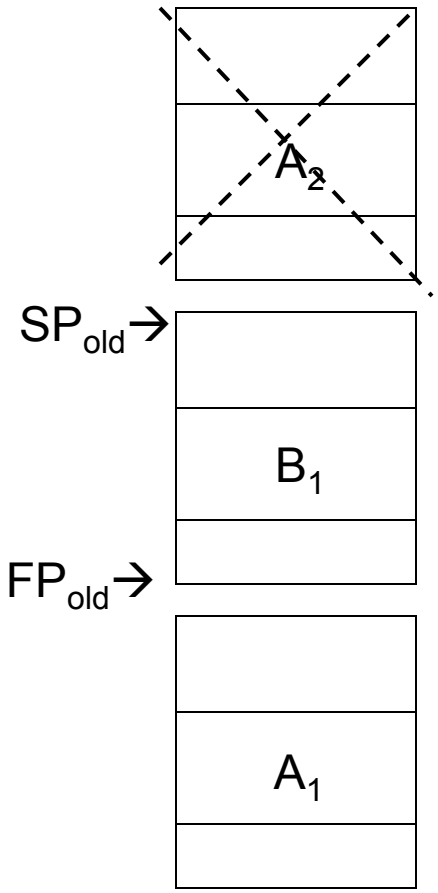
- allocate AR space
- Store current FP, SP there
- store arguments there
- set SP,FP to new values

Returning from call

```
proc A (aargs) {…B(bv)…}
proc B (bargs) {…A(av)…}
begin
   A (…);
end
```

Code for procedure must

- Restore previous FP, SP
  - these in AR for A
- deallocation of AR generally not necessary
  - accomplished by resetting FP,SP registers



Compiling procedure declarations

- Produces code for the procedure body
  - this “lives” at its own new address
  - quite similar to compiling straight-line Micro+ programs
- Main difference: de-referencing variables is (slightly) more complicated.
  - recall that a program variable in C is one of the following:
    - globally declared,
    - procedure argument, or
    - locally declared
      - in this case, handled just like in Micro+

Compiling procedure declarations

global-declarations;
int foo (char v) { *body-foo* }
void bar (int a) { *body-bar* }
…
void main (int argv[]) { *body-main* }

L_{foo} : <code for *body-foo* >

L_{bar} : <code for *body-bar* >

L_{main} : <code for *body-main* >

Here, the code for the procedure bodies is compiled just as Micro+, except:

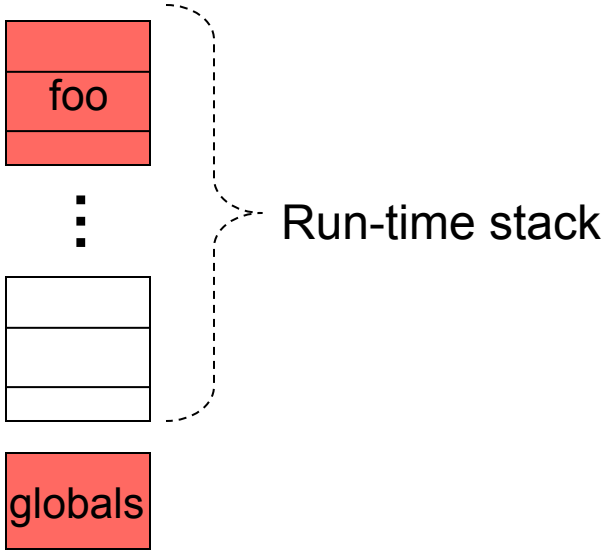
- There may be a "return" value. Usually passed in a designated register.
- and variable references

…as a consequence

global-declarations;
int foo (char v) { *decls-foo* … ***variable*** … }
void bar (int a) { … *body-bar* … }
…
void main (int argv[]) {… *body-main* … }

variable is stored **here**

…or **there**



Summary

- Without nesting, compiling procedure calls and definitions is straightforward
- For more, see pages 415-419 of “Compilers: Principles, Tools, and Techniques”
  - AKA “The Dragon Book” (1st Edition)
  - in 2nd edition, 430-435
    - Presentation in 1st edition is (alas) better