CS240: Programming in C

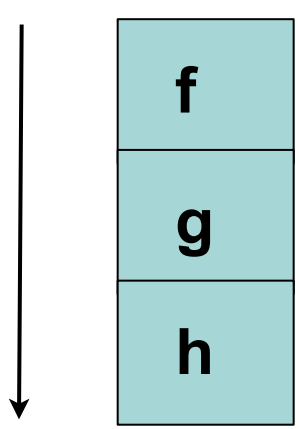
Lecture 15: Exception Handling

Exceptions

- An exception is a control-flow mechanism that allows unexpected (and perhaps erroneous) conditions to be caught and handled
- Key property: the handler for an exception need not be *lexically* apparent at the point where the exception is *raised*
- Concept: lexical vs. dynamic scope

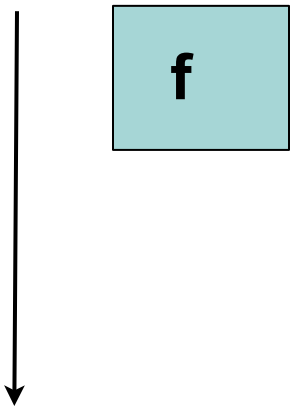
The Call Stack

Suppose function f calls function g which then calls function h



provide a handler for exception E

raise exception E:
control directly reverts to the handler defined in f



In Java ...

```
public void divide(int numberToDivide, int numberToDivideBy)
   throws BadNumberException{
       if(numberToDivideBy == 0){
           throw new BadNumberException("Cannot divide by 0");
       }
       return numberToDivide / numberToDivideBy;
   }
```

```
public void f(){
        try {
            int result = g(2,1);
            System.out.println(result);
        } catch (BadNumberException e) {
            //do something clever with the exception
            System.out.println(e.getMessage());
        }
        System.out.println("Division attempt done");
```

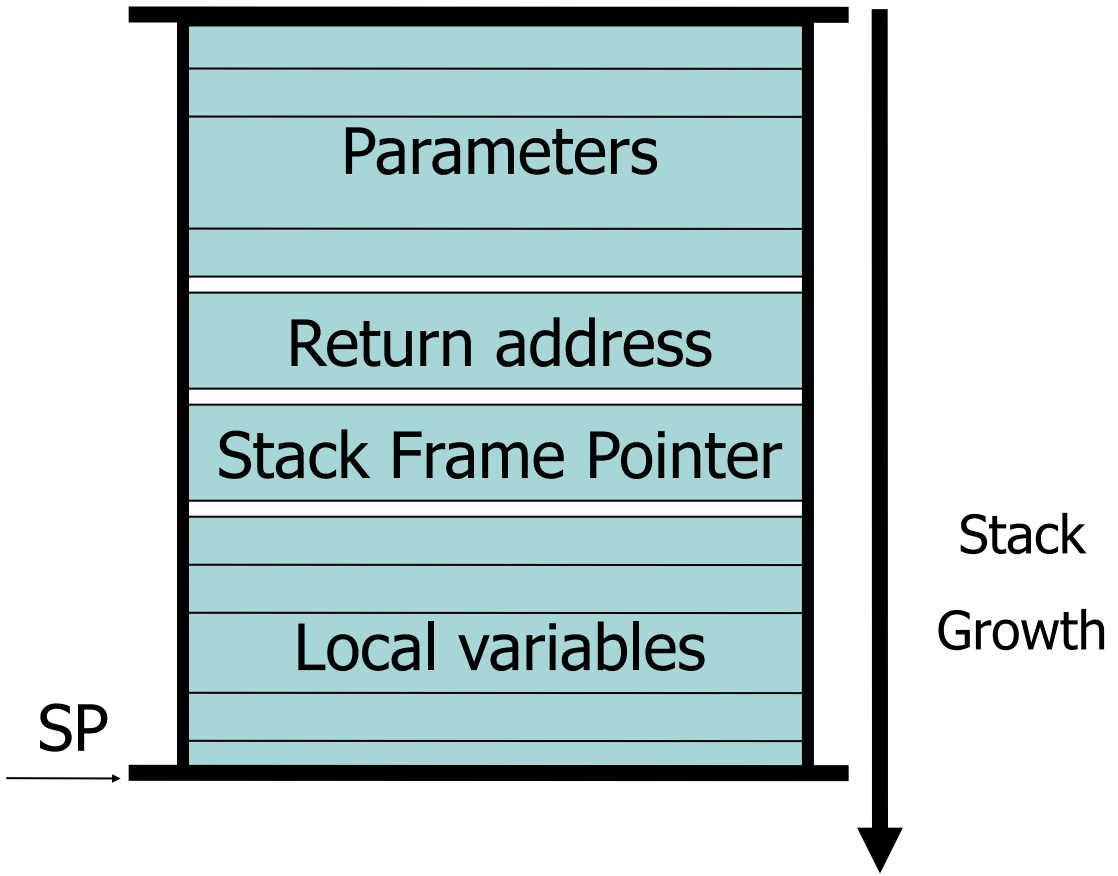
```
public void g(int x, y){
        ....
        r = divide(x,y);
        ...
}
```

What do we do in C?

- No primitive exception support in C
- Instead C provides two operations to effect non-local control-flow
  - save the set of all registers (setjmp)
  - jump to the saved set
- Since the set of registers includes the stack pointer, frame pointer, and program counter, we have a way of recovering back to any point in the execution.

Stack frame

- Parameters for the procedure
- Save current PC onto stack (return address)
- Save current SP value onto stack
- Allocates stack space for local variables by decrementing SP by appropriate amount



setjmp

```
#include < setjmp.h >
int setjmp(jmp_buf env);
```

The definition of setjmp expects an argument of type jmp_buf

```
#define _JBLEN  9
typedef struct { int _jb[_JBLEN + 1]; } jmp_buf[1];
```

Objects of type jmp_buf are defined to be arrays of length _JBLEN+1. Each element in this array will hold the contents of a specific register stored by setjmp

longjmp

```
longjmp(jmp_buf env, int val);
```

Restores the set of registers stored in env. But, this includes the program counter (pc). *So, this means longjmp does not return!*

The PC saved by setjmp points to the instruction that expects setjmp's return value. The second argument supplied to longjmp provides this value.

Example

```
#include < setjmp.h >

main()
{
  jmp_buf env;
  int i;

  i = setjmp(env);
  printf("i = %d\n", i);

  if (i != 0) exit(0);

  longjmp(env, 2);
  printf("This line does not get printed\n");

}
```

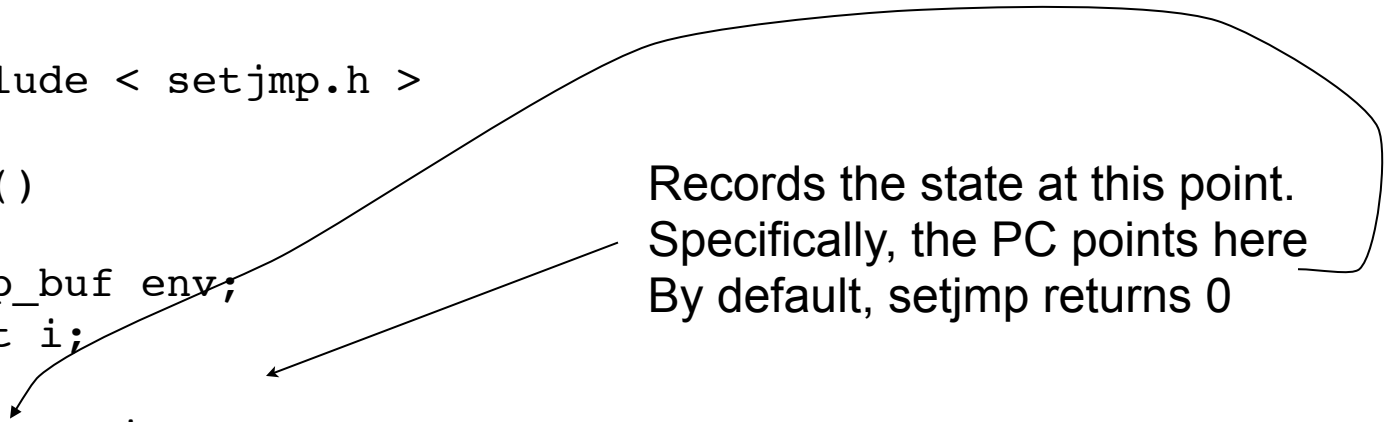
When run this prints:

i = 0

i = 2

Example

```
#include < setjmp.h >

main()
{
  jmp_buf env;
  int i;

  i = setjmp(env);
  printf("i = %d\n", i);

  if (i != 0) exit(0);

  longjmp(env, 2);
  printf("This line does not get printed\n");

}
```



When run this prints:

i = 0

i = 2

Example

```
#include < setjmp.h >

main()
{
   jmp_buf env;
   int i;

   i = setjmp(env);
   printf("i = %d\n", i);

   if (i != 0) exit(0);

   longjmp(env, 2);
   printf("This line does not get printed\n");

}
```

Records the state at this point. Specifically, the PC points here By default, setjmp returns 0

When we call longjmp, control jumps back to the point where the last setjmp was called, supplying 2 as the value assigned to i.

When run this prints:

i = 0

i = 2

Exceptions and Error Handling

```
jmp_buf jumper;

int SomeFunction(int a, int b)
   {
      if (b == 0) // can't divide by 0
         longjmp(jumper, -3);
      return a / b;
   }

void main(void)
   {
      if (setjmp(jumper) == 0)
      {
         int Result = SomeFunction(7, 0);
         // continue working with Result
      }
      else
         printf("an error occured\n");
   }
```

Example

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


int a(char *s, jmp_buf env)
{
  int i;

  i = setjmp(env);
  printf("Setjmp returned --
            i = %d, 0x%x\n", i, s);

  printf("s = %s\n", s);
  return i;
}
```

```
int b(int i, jmp_buf env)
{
  printf("In B: i=%d.
          Calling longjmp(env, i)\n",
i);

  longjmp(env, i);
}



int main()
{
  jmp_buf env;

  if (a("Me", env) != 0) exit(0);
  b(3, env);
}
```

What's wrong with this program?