CS 241 Lecture 23

Graham Cooper

July 27th, 2015

1 Tail Recursion in WLP4

```
int f(...) {
-- if(...){
-- -- if (..){
---- } else {}
-- }
-- else {}
return x;
}
```

Is the same as:

```
int f(...) {
-- if(...){
-- -- if (..){
---- } else {}
-- return x;
-- }
-- else {
-- return x;
--}
}
```

is the same as:

```
int f(...) {
-- if(...){
-- -- if (..){
-- -- return x;
-- -- }
-- -- else {
-- -- return x;
-- }
-- return x;
-- }
-- else {
-- return x;
--}
}
```

When return x follows an assignment to x, merge:
x = f(...) → return f(...)
return x;

- may create some tail recursive calls

Generalization:

- tail call optimization
- when a function's last action is any function call (recursive or not) can reuse the stack frame

Overloading

What would happen if we wanted to compile:

```
int f(int a){...}
int f(int a, int *b){...}
```

Get duplicated labels for f.
How do we fix this?

Name Mangling

Encode the types of params as part of the label

Example naming convention:
F + typeinfo + _ + name

ie.
1. int f(){...}
2. int f(int a){...}
3. int f(int a, int *b){...}

1. F_f:
2. Fi_f:
3. Fip_f:

- C++ compilers wil ldo this because c++ has overloading
- there is no standard mangling convention
- all compilers are different
- makes it hard or impossible to link code from different compilers
- this is by design b/c compilers differ in other aspects as well

C doesn't have overloading so there is no mangling
- C and C++ code call each other routinely
- How is this done?
- Suppress mangling in c++
Call C from C++
- Extern "C" int f(int n); tells c++ f wot be mangled

Call c++ from C - tell c++ not to mangle the function

extern "C" int g(int x){...} // dont mangle g

and then obviously you cannot overload extern c functions

Memory Management and the Heap

WLP4, C, C++

- explicit memory management
- user must free own data using free/delete

Java, Scheme

- implicit memory management
- garbage collection

How do new/delete or malloc/free work?

There are a variety of algorithms

1)

List of free blocks:

- maintain a linked list of ptrs to blocks of free RAM
- Initially entire heap is free, list contains one entry
- Suppose heap is 1k
- suppose we allocate 16 bytes
  - actually allocate 20 bytes + 1 int (4 bytes)
  - return pointer to second word
  - store size just before the returned pointer
  - free list ctonains the rest of the loop

Note: Repeated allocation and deallocation creates "holes" in the heap
EG:

```
alloc 20 {xx 20 xx,... (140)... ...}
alloc 40 {xx 20 xx, xx 40 xx,...(100) ... ...}
alloc 20 {...(20)..., xx 40 xx, ... (100)...}
alloc 5 {xx 5 xx, ... (15). .., xx 40 xx, ... (100)...}
etc.
```

We get holes like the 15 block hole on the last line, this causes:
Code fragmentation - means even if n bytes are free, we may not be able to allocate n bytes

To reduce fragmentation:
- don't always pick the first block of RAM big enought to satisfy the request