15213 Recitation Section C

Shimin Chen
Sept. 23, 2002

Outline

- Last week's exercise
- Function and stack
- Array
- Struct and linked-list

Last Week's Final Example

```
int  func5(int  x){  ???  }
0x80483c0     push %ebp
0x80483c1     mov  %esp,%ebp
0x80483c3     mov  0x8(%ebp),%ecx
0x80483c6     xor  %eax,%eax
0x80483c8     xor  %edx,%edx
0x80483ca     cmp  %ecx,%edx
0x80483cc     jge  0x80483d7
0x80483ce     mov  %esi,%esi
0x80483d0     add  %edx,%eax
0x80483d2     inc  %edx
0x80483d3     cmp  %ecx,%edx
0x80483d5     jl   0x80483d0
0x80483d7     mov  %ebp,%esp
0x80483d9     pop  %ebp
0x80483da     ret
```

Body

Write Comments

```
int  func5(int  x){  ???  }
0x80483c3   mov   0x8(%ebp),%ecx     ecx = x
0x80483c6   xor   %eax,%eax          eax = 0
0x80483c8   xor   %edx,%edx          edx = 0
0x80483ca   cmp   %ecx,%edx          if (edx>=x)
0x80483cc   jge   0x80483d7            goto L1
0x80483ce   mov   %esi,%esi          nop
0x80483d0   add   %edx,%eax          L2:eax += edx
0x80483d2   inc   %edx               edx ++
0x80483d3   cmp   %ecx,%edx          if (edx<x)
0x80483d5   jl    0x80483d0            goto L2
0x80483d7   ……                       L1:
```

Name the variables

- eax– result, edx--i

```
0x80483c3   mov   0x8(%ebp),%ecx     ecx = x;
0x80483c6   xor   %eax,%eax          result = 0;
0x80483c8   xor   %edx,%edx          i = 0;
0x80483ca   cmp   %ecx,%edx          if (i>=x)
0x80483cc   jge   0x80483d7            goto L1;
0x80483ce   mov   %esi,%esi
0x80483d0   add   %edx,%eax          L2:result += i;
0x80483d2   inc   %edx                  i++;
0x80483d3   cmp   %ecx,%edx          if (i<x)
0x80483d5   jl    0x80483d0            goto L2;
0x80483d7   ……                       L1:
```

Loop

```
result = 0;
i = 0;
if (i>=x)
  goto L1;

L2:result += i;
   i++;
if (i<x)
  goto L2;
L1:
```

```
result = 0; i = 0;
if (i>=x)  goto L1;
do {
    result += i;
    i++;
}while (i<x);
L1:
```

```
result = 0; i = 0;
While (i<x){
    result += i;
    i++;
}
```

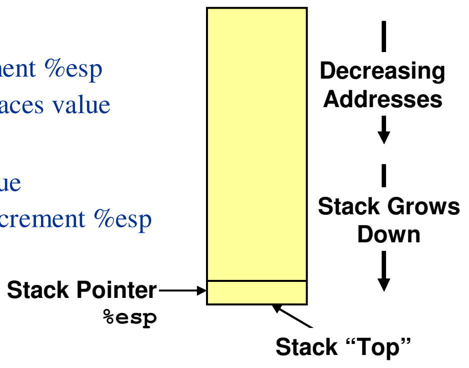
```
result = 0;
for (i=0; i<x; i++)
    result += i;
```

C Code

```
int func5(int x)
{
     int result=0;
     int i;
     for (i=0; i<x; i++)
        result += i;
     return result;
}
```

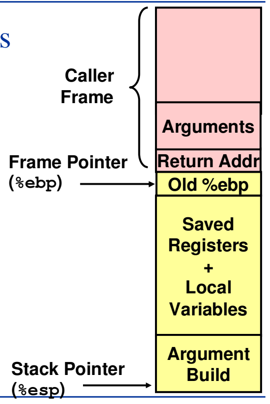
Stack Basics

- push
  - decrement %esp
  - then places value
- pop
  - get value
  - then increment %esp



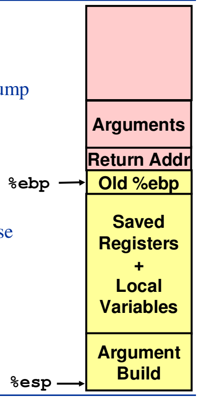
Function Stack Frames

- A caller function calls a callee function



Making a Call

- Caller:
  - "push" arguments (*in what order?*)
  - "call": put *return address* onto stack, jump to the start of callee function
- Callee:
  - save (caller's) %ebp
  - set up stack frame
  - save *callee-saved* registers if want to use
    - %ebx, %esi, %edi
  - put return value in %eax
  - restore %ebp and %esp
  - "ret" to jump to the "Return Addr"



Example 1

- Please draw the stack at the marked points
- Write C code for the assembly code
  - (gdb) x/s 0x8048478

  0x8048478 <_IO_stdin_used+4>: "%d\n"

```
int example_1 (int x, int y)
0x80483e4    push   %ebp
0x80483e5    mov    %esp,%ebp
0x80483e7    mov    0xc(%ebp),%eax
0x80483ea    add    0x8(%ebp),%eax
0x80483ed    mov    %ebp,%esp
0x80483ef    pop    %ebp
0x80483f0    ret
```

2.Stack?

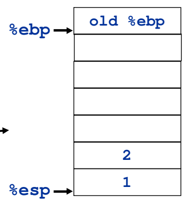
ASM of main()

```
0x80483f4    push   %ebp
0x80483f5    mov    %esp,%ebp
0x80483f7    sub    $0x8,%esp
0x80483fa    add    $0xfffffff8,%esp
0x80483fd    push   $0x2
0x80483ff    push   $0x1
0x8048401    call   0x80483e4 <example_1>
0x8048406    add    $0xfffffff8,%esp
0x8048409    push   %eax
0x804840a    push   $0x8048478
0x804840f    call   0x8048308 <printf>
0x8048414    xor    %eax,%eax
0x8048416    mov    %ebp,%esp
0x8048418    pop    %ebp
0x8048419    ret
```

1.Stack?
3.Stack?

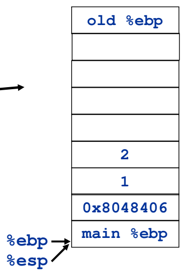
Stack at Point 1

```
<main>
0x80483f4   push   %ebp
0x80483f5   mov    %esp,%ebp
0x80483f7   sub    $0x8,%esp
0x80483fa   add    $0xfffffff8,%esp
0x80483fd   push   $0x2
0x80483ff   push   $0x1
0x8048401   call   0x80483e4
   <example_1>
0x8048406   ............
```



Stack at Point 2

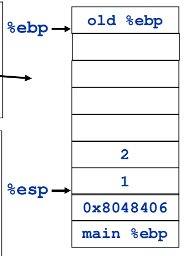
```
<example_2>
0x80483e4    push   %ebp
0x80483e5    mov    %esp,%ebp
0x80483e7    mov    0xc(%ebp),%eax
0x80483ea    add    0x8(%ebp),%eax
0x80483ed    mov    %ebp,%esp
0x80483ef    pop    %ebp
0x80483f0    ret
```



Stack at Point 3

```
<main>
0x80483ff    ………
0x8048401    call 0x80483e4 <example_1>
0x8048406    …………
```

```
<example_2>
0x80483e4    push   %ebp
0x80483e5    mov    %esp,%ebp
0x80483e7    mov    0xc(%ebp),%eax
0x80483ea    add    0x8(%ebp),%eax
0x80483ed    mov    %ebp,%esp
0x80483ef    pop    %ebp
0x80483f0    ret
```



Write Comments

int example_1 (int x, int y)

```
0x80483e4    push   %ebp
0x80483e5    mov    %esp,%ebp
0x80483e7    mov    0xc(%ebp),%eax
0x80483ea    add    0x8(%ebp),%eax
0x80483ed    mov    %ebp,%esp
0x80483ef    pop    %ebp
0x80483f0    ret
```

```
eax=y
eax+=x
```

main()

```
0x80483f4    push   %ebp
0x80483f5    mov    %esp,%ebp
0x80483f7    sub    $0x8,%esp
0x80483fa    add    $0xfffffff8,%esp
0x80483fd    push   $0x2
0x80483ff    push   $0x1
0x8048401    call   0x80483e4 <example_1>
0x8048406    add    $0xfffffff8,%esp
0x8048409    push   %eax
0x804840a    push   $0x8048478
0x804840f    call   0x8048308 <printf>
0x8048414    xor    %eax,%eax
0x8048416    mov    %ebp,%esp
0x8048418    pop    %ebp
0x8048419    ret
```

```
example_1(1,2)

printf("%d\n",
result_example_1)

return 0;
```

C Code

```
int example_1 (int x, int y)
{
        return x+y;
}

int main ()
{
        int result;

        result = example_1 (1, 2);
        printf ("%d\n", result);

        return 0;
}
```

Example 2: Recursion

- Please write C code for the assembly code
- Draw the stack changes of calling **`example_2(3)`**

`int example_2 (int x)`

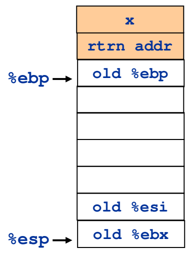
```
0x8048420   push   %ebp
0x8048421   mov    %esp,%ebp
0x8048423   sub    $0x10,%esp
0x8048426   push   %esi
0x8048427   push   %ebx
0x8048428   mov    0x8(%ebp),%ebx
0x804842b   cmp    $0x2,%ebx
0x804842e   jg     0x8048437
0x8048430   mov    $0x1,%eax
0x8048435   jmp    0x8048453
```

Example 2 Cont’d

```
0x8048437   add    $0xfffffff4,%esp
0x804843a   lea    0xfffffffe(%ebx),%eax
0x804843d   push   %eax
0x804843e   call   0x8048420 <example_2>
0x8048443   mov    %eax,%esi
0x8048445   add    $0xfffffff4,%esp
0x8048448   lea    0xffffffff(%ebx),%eax
0x804844b   push   %eax
0x804844c   call   0x8048420 <example_2>
0x8048451   add    %esi,%eax
0x8048453   lea    0xffffffe8(%ebp),%esp
0x8048456   pop    %ebx
0x8048457   pop    %esi
0x8048458   mov    %ebp,%esp
0x804845a   pop    %ebp
0x804845b   ret
```

Stack Frame

```
<example_2>
0x8048420   push   %ebp
0x8048421   mov    %esp,%ebp
0x8048423   sub    $0x10,%esp
0x8048426   push   %esi
0x8048427   push   %ebx

. . . . . .

0x8048453   lea    0xffffffe8(%ebp),%esp
0x8048456   pop    %ebx
0x8048457   pop    %esi
0x8048458   mov    %ebp,%esp
0x804845a   pop    %ebp
0x804845b   ret
```



Write Comments For Body

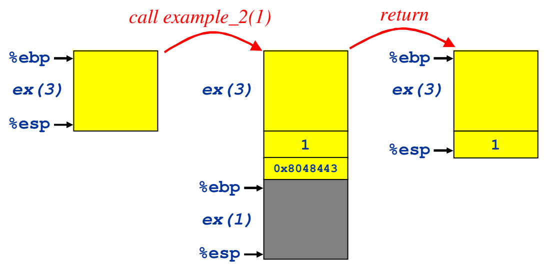
```
0x8048428   mov    0x8(%ebp),%ebx              ebx=x
0x804842b   cmp    $0x2,%ebx                   if (x>2)
0x804842e   jg     0x8048437                     goto L1
0x8048430   mov    $0x1,%eax                   eax=1
0x8048435   jmp    0x8048453                   goto L2
0x8048437   add    $0xfffffff4,%esp            L1:
0x804843a   lea    0xfffffffe(%ebx),%eax
0x804843d   push   %eax                        push x-2
0x804843e   call   0x8048420 <example_2>       example_2
0x8048443   mov    %eax,%esi                   esi=eax
0x8048445   add    $0xfffffff4,%esp
0x8048448   lea    0xffffffff(%ebx),%eax
0x804844b   push   %eax                        push x-1
0x804844c   call   0x8048420 <example_2>       example_2
0x8048451   add    %esi,%eax                   eax+=esi
0x8048453   . . .                              L2:
```

C Code

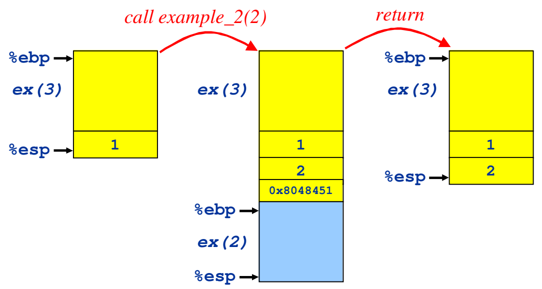
```
int example_2 (int n)
{int result;

   if (n <= 2)
     result = 1;
   else
     result = example_2(n-2)
            + example_2(n-1);

   return result;
}
```

Fibonacci Numbers

Stack Changes of example_2(3)



Stack Changes of example_2(3)



Arrays

- Allocated as contiguous blocks of memory
- Address Computation Example

int cmu[5];	/* at address 'addr' */
cmu[0]	addr+0
cmu[3]	addr+3*sizeof(int)
cmu[-1]	addr+(-1)*sizeof(int)

Example 3 Write C Code

```
0x80483f0   push   %ebp
0x80483f1   mov    %esp,%ebp
0x80483f3   push   %ebx
0x80483f4   mov    0x8(%ebp),%ebx
0x80483f7   mov    0xc(%ebp),%ecx
0x80483fa   xor    %eax,%eax
0x80483fc   xor    %edx,%edx
0x80483fe   cmp    %ecx,%eax
0x8048400   jge    0x804840a
0x8048402   add    (%ebx,%edx,4),%eax
0x8048405   inc    %edx
0x8048406   cmp    %ecx,%edx
0x8048408   jl     0x8048402
0x804840a   pop    %ebx
0x804840b   mov    %ebp,%esp
0x804840d   pop    %ebp
0x804840e   ret
```

Write Comments

```
0x80483f0   push %ebp
0x80483f1   mov  %esp,%ebp
0x80483f3   push %ebx
0x80483f4   mov  0x8(%ebp),%ebx
0x80483f7   mov  0xc(%ebp),%ecx
0x80483fa   xor  %eax,%eax
0x80483fc   xor  %edx,%edx
0x80483fe   cmp  %ecx,%eax
0x8048400   jge  0x804840a
0x8048402   add  (%ebx,%edx,4),%eax
0x8048405   inc  %edx
0x8048406   cmp  %ecx,%edx
0x8048408   jl   0x8048402
0x804840a   pop  %ebx
0x804840b   mov  %ebp,%esp
0x804840d   pop  %ebp
0x804840e   ret
```

```
ebx=arg1
ecx=arg2
eax=0
edx=0
if (0>=arg2)
  goto L1
L2:eax+=arg1[edx]
edx++
if (edx<arg2)
  goto L2
L1:
```

Write Comments

```
0x80483f0   push %ebp
0x80483f1   mov  %esp,%ebp
0x80483f3   push %ebx
0x80483f4   mov  0x8(%ebp),%ebx
0x80483f7   mov  0xc(%ebp),%ecx
0x80483fa   xor  %eax,%eax
0x80483fc   xor  %edx,%edx
0x80483fe   cmp  %ecx,%eax
0x8048400   jge  0x804840a
0x8048402   add  (%ebx,%edx,4),%eax
0x8048405   inc  %edx
0x8048406   cmp  %ecx,%edx
0x8048408   jl   0x8048402
0x804840a   pop  %ebx
0x804840b   mov  %ebp,%esp
0x804840d   pop  %ebp
0x804840e   ret
```

```
arg1:x, arg2:num
edx:i, eax:result
```

```
ebx=x
ecx=num
result=0
i=0
if (0>=num)
  goto L1
L2:result+=x[i]
i++
if (i<num)
  goto L2
L1:
```

Loop + Array

```
int example_3 (int x[], int num)
{
   int i, result;

   result = 0;
   for (i=0; i<num; i++)
      result += x[i];

   return result;
}
```

Struct and Linked List

- struct a_struct {
      int   a;
      float b;
      char   c[20];
  };
- struct b_struct {
      . . .
      struct b_struct *link;
      . . .
  };

Example 4: Write C Code

```
0x8048434   push   %ebp
0x8048435   mov    %esp,%ebp
0x8048437   mov    0x8(%ebp),%edx
0x804843a   xor    %eax,%eax
0x804843c   test   %edx,%edx
0x804843e   je     0x8048449
0x8048440   add    0x4(%edx),%eax
0x8048443   mov    (%edx),%edx
0x8048445   test   %edx,%edx
0x8048447   jne    0x8048440
0x8048449   mov    %ebp,%esp
0x804844b   pop    %ebp
0x804844c   ret
```

Hint: the code visits a linked list

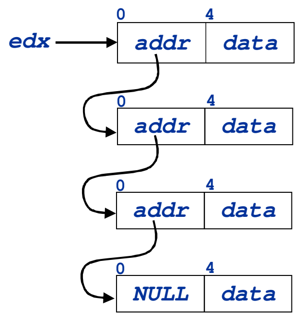
Write Comments

```
0x8048434   push %ebp
0x8048435   mov  %esp,%ebp
0x8048437   mov  0x8(%ebp),%edx
0x804843a   xor  %eax,%eax
0x804843c   test %edx,%edx
0x804843e   je   0x8048449
0x8048440   add  0x4(%edx),%eax
0x8048443   mov  (%edx),%edx
0x8048445   test %edx,%edx
0x8048447   jne  0x8048440
0x8048449   mov  %ebp,%esp
0x804844b   pop  %ebp
0x804844c   ret
```

```
edx=arg1
eax=0
if (edx == 0)
  goto L1
L2:eax += *(edx+4)
edx = *(edx)
if (edx != 0)
  goto L2
L1:
```

Understand the Loop

- *edx is an address*



```
edx=arg1
eax=0
if (edx == 0)
  goto L1
L2:eax += *(edx+4)
edx = *(edx)
if (edx != 0)
  goto L2
L1:
```

Name the Variables

- *arg1: head*
- *eax: result*
- *edx: p*
- **(edx+4): p->data*
- **(edx): p->next*
- *NULL is 0*

```
edx=arg1
eax=0
if (edx == 0)
  goto L1
L2:eax += *(edx+4)
edx = *(edx)
if (edx != 0)
  goto L2
L1:
```

Name the Variables

- *arg1: head*
- *eax: result*
- *edx: p*
- **(edx+4): p->data*
- **(edx): p->next*
- *NULL is 0*

```
p=head
result=0
if (p == NULL)
  goto L1
L2:result += p->data
p = p->next
if (p != NULL)
  goto L2
L1:
```

C Code

```
Struct linked_list {
  struct linked_list *next;
  int                 data;
};

int example_4 (struct linked_list *head)
{
   int result;
   struct linked_list *p;

   result = 0; p = head;
   while (p != NULL) {
      result += p->data;
      p = p->next;
   }
   return result;
}
```