**Recursive procedure**

```
int rfact(int n) {  
  int result;  
  if (n <=1)  
    result = 1;  
  else  
    result = n * rfact(n-1);  
  return result;  }
```

“multiple of 16 bytes”

x86 programming guideline; including 4 bytes for the old %ebp and 4 bytes for the return address, caller uses 32 bytes; alignment issues (3.9.3)

### Stack Comment

<table>
<thead>
<tr>
<th>addr</th>
<th>Stack</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>%esp</td>
<td>n = 3</td>
<td>caller</td>
</tr>
<tr>
<td>%esp</td>
<td>return addr</td>
<td>caller</td>
</tr>
<tr>
<td>%esp %ebp</td>
<td>%ebp</td>
<td>Caller value</td>
</tr>
<tr>
<td>%esp</td>
<td>%ebx</td>
<td>Caller value</td>
</tr>
<tr>
<td>%esp</td>
<td>-4 to -16</td>
<td>unused</td>
</tr>
</tbody>
</table>
| %esp | -20: 2 | %ebx=3 
%eax=1,2 |
| %esp | return address | rfact |
| %esp %ebp | %ebp | rfact value |
| %esp | %ebx = 3 | unused |
| %esp | -4 to -16 | unused |
| %esp | -20: 1 | %ebx=2 
%eax=1,1 |
| %esp | return address | rfact |
| %esp %ebp | %ebp | rfact value |
| %esp | %ebx = 2 | unused |
| %esp | -4 to -16 | unused |
| %esp | -20: | %ebx=1 
%eax=1 |
| | | jle .L3 |

### POPPING:

%ebx = 2, 3  
%eax = 1, 2, 6

---

**CALL** ➔ Pushes the return address onto the stack (%esp-4 and mov);  
**RETURN** ➔ pops it

---

rfact:

```
pushl %ebp  
movl %esp, %ebp  
pushl %ebx  
subl $20, %esp  
movl 8(%ebp), %ebx  
movl $1, %eax  
cmpl $1, %ebx  
jle .L3  
leal -1(%ebx), %eax  
movl %eax, (%esp)  
call rfact  
imull %ebx, %eax  
.L3:  
addl $20, %esp  
popl %ebx  
popl %ebp  
ret
```
Array allocation and access

- **type array[length]**
  - Contiguously *allocated* region of length * sizeof(T) * bytes
  - Starting location of array is a pointer (x)
  - **Access** array elements using integer index i ranging between 0 and length-1 (i.e. the subscript)
    - Array element i will be stored at address \( x + \text{sizeof}(T) \times i \)

Total size: 12, 20, & 32
Element i:
- \( x + 1 \times i \)
- \( x + 4 \times i \)
- \( x + 8 \times i \)

Address of array in %edx and i stored in %ecx
- \textbf{movl} (%edx,%ecx,4)
Array allocation and access (cont)

- Explains why scaled factors are 1, 2, 4, and 8
  - The primitive data types
- Problem 3.35 (pg 233)
- IA32
  - A pointer of any kind is 4 bytes long
  - GCC allocates 12 bytes for the data type long double
    - 4 bytes for float and pointers, 8 bytes for double, 12 bytes for long double

---

<table>
<thead>
<tr>
<th>Given</th>
<th>Array</th>
<th>Element size</th>
<th>Total Size</th>
<th>Start address</th>
<th>Element i</th>
</tr>
</thead>
<tbody>
<tr>
<td>short S[7]</td>
<td>S</td>
<td>2</td>
<td>14</td>
<td>x_s</td>
<td>x_s + 2i</td>
</tr>
<tr>
<td>short *T[3]</td>
<td>T</td>
<td>4</td>
<td>12</td>
<td>x_t</td>
<td>x_t + 4i</td>
</tr>
<tr>
<td>long double V[8]</td>
<td>V</td>
<td>12</td>
<td>96</td>
<td>x_v</td>
<td>x_v + 12i</td>
</tr>
<tr>
<td>long double *W[4]</td>
<td>W</td>
<td>4</td>
<td>16</td>
<td>x_w</td>
<td>x_w + 4i</td>
</tr>
</tbody>
</table>
## Pointer arithmetic

### Reminders...

- C allows arithmetic on pointers, where the computed value is scaled according to the size of the data type referenced by the pointer
  - So, if p is a pointer to data type T
  - And, the value of p is \( x_p \)
  - Then, then \( p+i \) has value \( x_p + L \times i \)
  - Where, \( L \) is the size of data type T
  - Thus \( A[i] == \ast(A+i) \)

### Example

- \%edx → starting address of array E
- \%ecx → integer index \( i \)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Value</th>
<th>Assembly code... result in %eax</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E )</td>
<td>int *</td>
<td>( x_e )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( E[2] )</td>
<td>int</td>
<td>( M[x_e] )</td>
<td></td>
<td>Reference memory</td>
</tr>
<tr>
<td>( E[i] )</td>
<td>int</td>
<td>( M[x_e + 4i] )</td>
<td></td>
<td>Reference memory</td>
</tr>
<tr>
<td>&amp;( E[2] )</td>
<td>int *</td>
<td>( x_e + 8 )</td>
<td></td>
<td>Generate address</td>
</tr>
<tr>
<td>( E+i-1 )</td>
<td>int *</td>
<td>( x_e + 4i - 4 )</td>
<td></td>
<td>Generate address</td>
</tr>
<tr>
<td>( *(E+i-3) )</td>
<td>int *</td>
<td>( M[x_e + 4i -12] )</td>
<td></td>
<td>Reference memory</td>
</tr>
<tr>
<td>&amp;( E[i]-E )</td>
<td>int</td>
<td>( i )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>