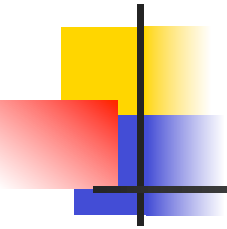
3.8 - 3.12 Summary

Alexandre David



3.8.4 & 3.8.5

- Array accesses

N X N Matrix Code

- **Fixed dimensions**
  - Know value of N at compile time

- **Variable dimensions, explicit indexing**
  - Traditional way to implement dynamic arrays

- **Variable dimensions, implicit indexing**
  - Now supported by gcc

```
#define N 16
typedef int fix_matrix[N][N];
/* Get element a[i][j] */
int fix_ele
  (fix_matrix a, int i, int j)
{
  return a[i][j];
}
```

```
#define IDX(n, i, j) ((i)*(n)+(j))
/* Get element a[i][j] */
int vec_ele
 (int n, int *a, int i, int j)
{
  return a[IDX(n,i,j)];
}
```

```
/* Get element a[i][j] */
int var_ele
 (int n, int a[n][n], int i, int j)
{
  return a[i][j];
}
```

16 X 16 Matrix Access

- **Array Elements**
  - Address **`A`** **`+`** $i * (C * K) + j * K$
  - C = 16, K = 4

```
/* Get element a[i][j] */
int fix_ele(fix_matrix a, int i, int j) {
  return a[i][j];
}
```

```
  movl   12(%ebp), %edx     # i
  sall   $6, %edx           # i*64
  movl   16(%ebp), %eax     # j
  sall   $2, %eax           # j*4
  addl   8(%ebp), %eax      # a + j*4
  movl   (%eax,%edx), %eax  # *(a + j*4 + i*64)
```

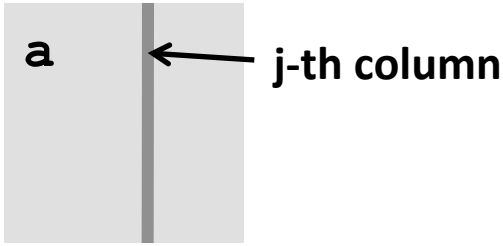
n X n Matrix Access

- **Array Elements**
  - Address **A** + *i* * (*C* * *K*) + *j* * *K*
  - C = n, K = 4

```
/* Get element a[i][j] */
int var_ele(int n, int a[n][n], int i, int j) {
  return a[i][j];
}
```

```
  movl   8(%ebp), %eax      # n
  sall   $2, %eax           # n*4
  movl   %eax, %edx         # n*4
  imull  16(%ebp), %edx     # i*n*4
  movl   20(%ebp), %eax     # j
  sall   $2, %eax           # j*4
  addl   12(%ebp), %eax     # a + j*4
  movl   (%eax,%edx), %eax  # *(a + j*4 + i*n*4)
```

Optimizing Fixed Array Access




```
#define N 16
typedef int fix_matrix[N][N];
```

- **Computation**
  - Step through all elements in column j
- **Optimization**
  - Retrieving successive elements from single column

```
/* Retrieve column j from array */
void fix_column
  (fix_matrix a, int j, int *dest)
{
  int i;
  for (i = 0; i < N; i++)
    dest[i] = a[i][j];
}
```

Optimizing Fixed Array Access

- **Optimization**
  - Compute ajp = &a[i][j]
    - Initially = a + 4*j
    - Increment by 4*N

Register	Value
%ecx	ajp
%ebx	dest
%edx	i

```
/* Retrieve column j from array */
void fix_column
  (fix_matrix a, int j, int *dest)
{
  int i;
  for (i = 0; i < N; i++)
    dest[i] = a[i][j];
}
```

```
.L8:                            # loop:
   movl  (%ecx), %eax           #   Read *ajp
   movl  %eax, (%ebx,%edx,4)    #   Save in dest[i]
   addl  $1, %edx               #   i++
   addl  $64, %ecx              #   ajp += 4*N
   cmpl  $16, %edx              #   i:N
   jne   .L8                    #   if !=, goto loop
```

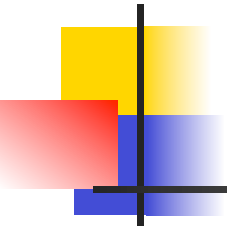
Optimizing Variable Array Access

- Compute ajp = &a[i][j]
  - Initially = a + 4*j
  - Increment by 4*n

Register	Value
%ecx	ajp
%edi	dest
%edx	i
%ebx	4*n
%esi	n

```
/* Retrieve column j from array */
void var_column
   (int n, int a[n][n],
    int j, int *dest)
{
   int i;
   for (i = 0; i < n; i++)
     dest[i] = a[i][j];
}
```

```
.L18:                              # loop:
   movl  (%ecx), %eax              #   Read *ajp
   movl  %eax, (%edi,%edx,4)       #   Save in dest[i]
   addl  $1, %edx                  #   i++
   addl  $ebx, %ecx                #   ajp += 4*n
   cmpl  $edx, %esi                #   n:i
   jg    .L18                      #   if >, goto loop
```

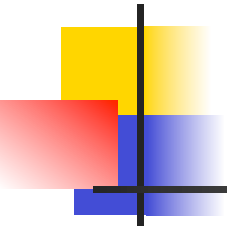


3.10

- Recap on pointers

3.11 gdb

- Debugger – important tool.
  - gdb: low level tool, important to know what it can do
  - Use a higher level tool based on gdb = front-end to gdb, e.g., kdbg.



3.12.1 Protection Against Attacks

- Some protections against buffer overflow
  - stack randomization
  - detection of stack corruption
  - (processor) execution flag

System-Level Protections

- **Randomized stack offsets**
  - At start of program, allocate random amount of space on stack
  - Makes it difficult for hacker to predict beginning of inserted code

- **Nonexecutable code segments**
  - In traditional x86, can mark region of memory as either "read-only" or "writeable"
    - Can execute anything readable
  - X86-64 added explicit "execute" permission

```
unix> gdb bufdemo
(gdb) break echo

(gdb) run
(gdb) print /x $ebp
$1 = 0xffffc638

(gdb) run
(gdb) print /x $ebp
$2 = 0xffffbb08

(gdb) run
(gdb) print /x $ebp
$3 = 0xffffc6a8
```

Stack Canaries

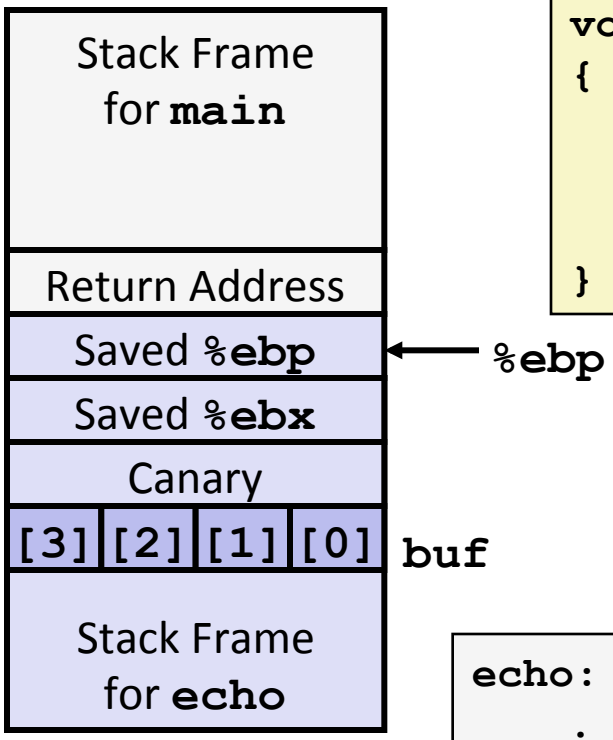
- **Idea**
  - Place special value ("canary") on stack just beyond buffer
  - Check for corruption before exiting function
- **GCC Implementation**
  - `-fstack-protector`
  - `-fstack-protector-all`

```
unix>./bufdemo-protected
Type a string:1234
1234
```

```
unix>./bufdemo-protected
Type a string:12345
*** stack smashing detected ***
```

Setting Up Canary

Before call to gets

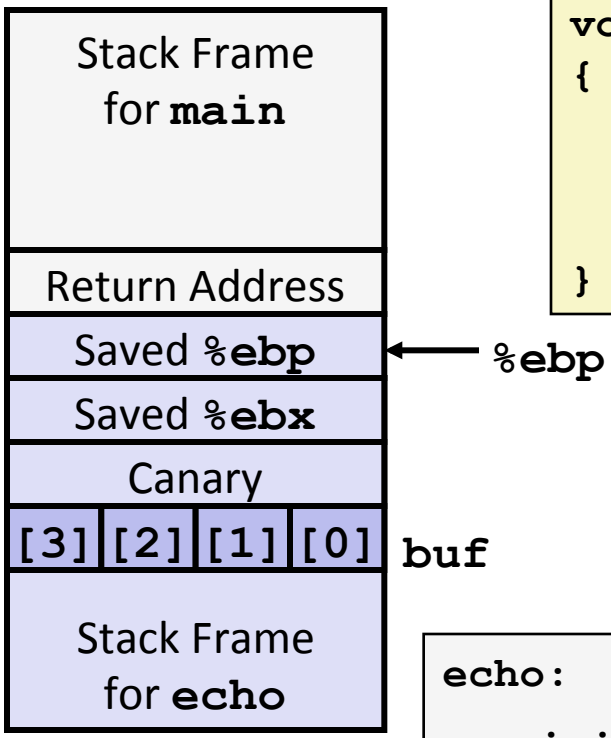



```
/* Echo Line */
void echo()
{
    char buf[4];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```

```
echo:
    . . .
    movl    %gs:20, %eax     # Get canary
    movl    %eax, -8(%ebp)   # Put on stack
    xorl    %eax, %eax       # Erase canary
    . . .
```

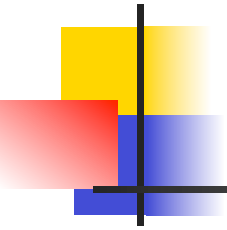
Checking Canary

Before call to gets



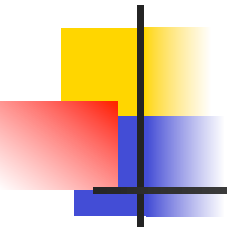

```
/* Echo Line */
void echo()
{
    char buf[4];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```

```
echo:
    . . .
    movl    -8(%ebp), %eax    # Retrieve from stack
    xorl    %gs:20, %eax      # Compare with Canary
    je      .L24              # Same: skip ahead
    call    __stack_chk_fail  # ERROR
.L24:
    . . .
```



3.13.1

- History, for your culture.



3.14

- “x87” FP
  - Old & cumbersome (stack).
- SSE
  - Set of SIMD instructions, support FP operations, better.