Memory-Related Perils and Pitfalls

- Dereferencing bad pointers
- Reading uninitialized memory
- Overwriting memory
- Referencing nonexistent variables
- Freeing blocks m
- Referencing free
- Failing to free blocks

C operators

Operators	*Associativity*
`() [] -> .`	left to right
`! ~ ++ -- + - * & (type) sizeof`	right to left
`* / %`	left to right
`+ -`	left to right
`<< >>`	left to right
`< <= > >=`	left to right
`== !=`	left to right
`&`	left to right
`^`	t to right
`|`	t to right
`&&`	t to right
`||`	left to right
`?:`	right to left
`= += -= *= /= %= &= ^= != <<= >>=`	right to left
`,`	left to right

- `->`, `()`, and `[]` have high precedence, with `*` and `&` just below
- Unary `+`, `-`, and `*` have higher precedence than binary forms

Source: K&R page 53

C Pointer Declarations: Test Yourself!

`int *p`	p is a pointer to int
`int *p[13]`	p is an array[13] of pointer to int
`int *(p[13])`	p is an array[13] of pointer to int
`int **p`	p is a pointer to a pointer to an int
`int (*p)[13]`	[13] of int
`int *f()`	f is a functi… pointer to int
`int (*f)()`	f is a pointer to a function returning int

Source: K&R Sec 5.12

Dereferencing Bad Pointers

- The classic `scanf` bug

```
int val;

...

scanf("%d", v
```

Reading Uninitialized Memory

- Assuming that heap data is initialized to zero

```
/* return y = Ax */
int *matvec(int **A, int *x) {
    int *y = malloc(N*sizeof(int));
    int i, j;

    for (i=0; i<N; i++)
        for (j=0; j<N; j++)
            y[i] += A[i][j]*x
    return y;
}
```

Overwriting Memory

- Allocating the (possibly) wrong sized object

```
int **p;

p = malloc(N*

for (i=0; i<N
   p[i] = malloc(M*sizeof(
}
```

Overwriting Memory

- Off-by-one error

```
int **p;

p = malloc(N*sizeof(int *));

for (i=0; i<=
   p[i] = malloc(M*sizeof
}
```

Overwriting Memory

- Not checking the max string size

```
char s[8];
int i;

gets(s);  /*                              m stdin */
```


- Basis for classic buffer overflow

Overwriting Memory

- Misunderstanding pointer arithmetic

```
int *search(int *p, int val) {

   while (*p && *p != val)
      p += si

   return p;
}
```