

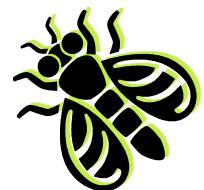
Debugging

Nigel Horspool
(with some material by Mike Zastre)



Topics

- Kinds of Programming Errors
- What is a Bug?
- Classifying Different Kinds of Bugs
- Bug Avoidance Strategies



History Lesson: The First Bug?

A dead moth found
trapped between
electrical contacts in
the **Mark II Aiken
Relay Calculator**,
9 September 1945.

(Some say it was 1947.)

Source: Wikipedia entry for
software bug

92

Photo # NH 96566-KN First Computer "Bug", 1945

9/9

0800 Antran started
1000 " stopped - antran ✓ { 1.2700 9.037 847 025
13^{sec} (032) MP - MC 1.1821 44000 9.037 846 995 contact
(033) PRO 2 2.130476415
contact 2.130676415
Relays 6-2 in 033 failed switch speed test
in relay " 10.00 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Multi+ Adder Test.
1545  Relay #70 Panel F
(Moth) in relay.
First actual case of bug being found.
1600 Antran started.
1700 closed down.

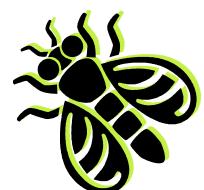
Relay 2145
Relay 3371



A Bug is NOT:

any kind of
mistake which
causes the
compiler to
generate an
error message

any kind of error
which causes
the linker to
report a problem
(typically an
undefined or
duplicate symbol).



Bug Terminology

failure

something visible to the program's user

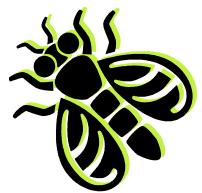
fault

the state of the program which leads to a failure

error

the incorrect code fragment which leads to the fault and thus to the failure

Source: Andrew Ko and Brad Myers, "Development and Evaluation of a Model of Programming Errors", HCC'03.



Some Observable Failures

- **Infinite Loop**
- **Deadlock**
- **Incorrect Output**
- **Memory Leak**
- **Run-Time Exception**
 - the exception possibilities are very language and platform dependent



Run-Time Failure Examples

Some Possibilities for C/C++ or Assembler Code

- “Blue Screen of Death”
- Segmentation fault
- Unaligned memory address
- Invalid instruction
- ...

The Infamous Blue Screen of Death

Rapid

A fatal exception 0E has occurred at 0028:C0011E36 in UXD UMM(01) +
00010E36. The current application will be terminated.

- * Press any key to terminate the current application.
- * Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue _

Now just a
memory!

But we have new BSOD flavours

You need to restart your computer. Hold down the Power button for several seconds or press the Restart button.

Veuillez redémarrer votre ordinateur. Maintenez la touche de démarrage enfoncée pendant plusieurs secondes ou bien appuyez sur le bouton de réinitialisation.

Sie müssen Ihren Computer neu starten. Halten Sie dazu die Einschalttaste einige Sekunden gedrückt oder drücken Sie die Neustart-Taste.

コンピュータを再起動する必要があります。パワー ボタンを数秒間押し続けるか、リセット ボタンを押してください。



Your PC ran into a problem that it couldn't handle, and now it needs to restart.

You can search for the error online: DPC WATCHDOG VIOLATION

It'll restart in: 9 seconds

```
Call Trace:
[<c041b7f2>] iounmap+0x9e/0xc8
[<c053480d>] agp_generic_free_gatt_table+0x2e/0x9e
[<c0533991>] agp_add_bridge+0x1a8/0x26f
[<c05439eb>] __driver_attach+0x0/0x6b
[<c04e6bf4>] pci_device_probe+0x36/0x57
[<c0543945>] driver_probe_device+0x42/0x8b
[<c0543a2f>] __driver_attach+0x41/0x6b
[<c054344a>] bus_for_each_dev+0x37/0x59
[<c05438af>] driver_attach+0x11/0x13
[<c05439eb>] __driver_attach+0x0/0x6b
[<c0543152>] bus_add_driver+0x64/0xfd
[<c04e6d22>] __pci_register_driver+0x47/0x63
[<c040044d>] init+0x17d/0x2f7
[<c0403dee>] ret_from_fork+0x6/0x1c
[<c04002d0>] init+0x0/0x2f7
[<c04002d0>] init+0x0/0x2f7
[<c0404c3b>] kernel_thread_helper+0x7/0x10
=====
Code: 78 29 8b 44 24 04 29 d0 8b 54 24 10 c1 f8 05 c1 e0 0c 09 f8 89 02 8b 43 0c
85 c0 75 06 0f 0b 9c 00 77 c8 61 c0 48 89 43 0c eb 08 <0f> 0b 9f 00 77 c8 61 c0
8b 03 f6 c4 04 0f 85 a5 00 00 00 a1 0c
EIP: [<c041b749>] change_page_attr+0x19a/0x275 SS:ESP 0068:c14f7ec0
<0>Kernel panic - not syncing: Fatal exception
```



Segmentation Fault

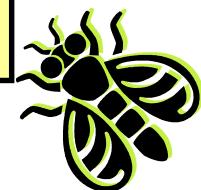
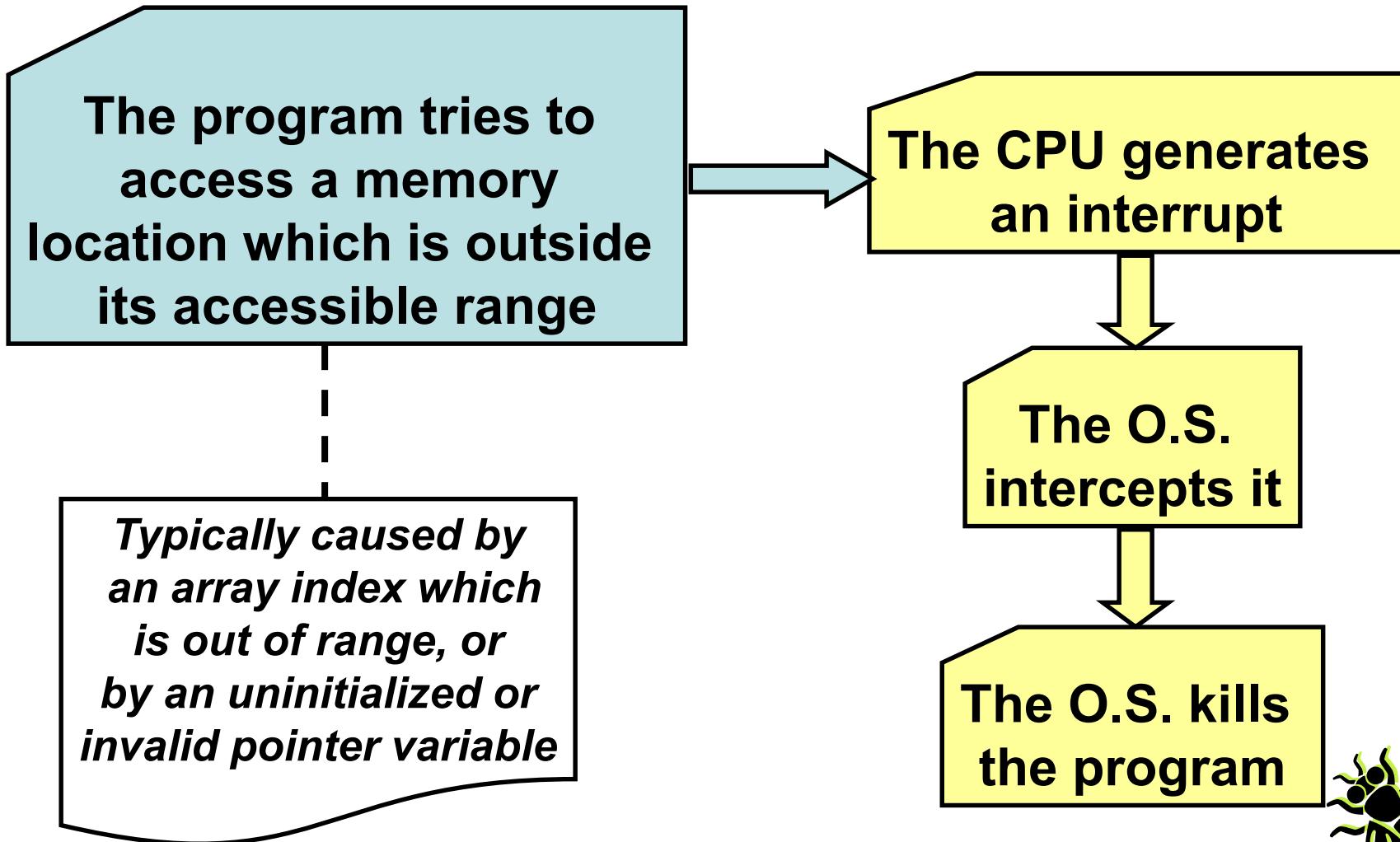
What happens if we try to run this incorrect program?

```
% gcc segfault.c  
% a.out  
* i = 0  
* i = 1  
Segmentation Fault  
(core dumped)  
%
```

```
#include <stdio.h>  
  
int A[100];  
  
int main() {  
    int *p, i;  
    p = A; i = 0;  
    while(1) {  
        printf("* i = %d\n", i++);  
        *p = 0;  
        p -= 100;  
    }  
    return 0;  
}
```



What is a Segmentation Fault?



Bus Error

```
# On Intel x86  
% gcc unaligned.c  
% a.out  
Storing 0 at address  
    80608A1
```

```
# On PowerPC  
% gcc unaligned.c  
% a.out  
Storing 0 at address 20901  
Bus Error  
%
```

```
/* C code example */  
#include <stdio.h>  
int A[100];  
  
int main() {  
    int *p, i;  
    p = A;  
    p = (int*)((int)A + 1);  
    printf("Storing 0 at"  
        " address %X\n", p);  
    *p = 0;  
    return 0;  
}
```



What is a Bus Error?

The program has tried
to access a
misaligned data item

e.g. a 4-byte integer
at a location which
is not divisible by 4.

*Typically caused
by uninitialized pointers*

Some computers
generate
an interrupt

Other computers
permit an unaligned
data access (slowly)
and continue



Illegal Instruction (sometimes)

```
# On Intel x86
# But depends on version
# of gcc
% gcc badinstr.c
% a.out
In foo
Illegal instruction
(core dumped)
%
```

```
/* C code example */
#include <stdio.h>

void foo() {
    int a, *ap;
    ap = &a;
    printf("In foo\n");
    ap[2] -= 3;
}

int main() {
    foo();
    printf("I'm back!\n");
    return 0;
}
```



How did we get to an Illegal Instruction?

The program has tried to execute an unassigned or privileged opcode

The CPU generates an interrupt

The O.S. intercepts it

The O.S. kills the program

caused either by overwriting the text (instruction) area or by overwriting the return address from a subroutine – and that's usually due to a bad array index or pointer.



Memory Leak

The program runs, getting slower and slower, until some random failure occurs.

In a driver program or similar, it's the OS which gradually degrades.

```
#include <stdlib.h>

void foo() {
    int *ap;
    ap = malloc(4000);
    ...
    // return without
    // freeing the array
}

int main() {
    ...
    foo();
    ...
}
```



Avoiding Memory Leaks

- Use a programming language where garbage collection is implicit (Java, C#, Perl, Python ...)
- Otherwise, have every subroutine deallocate the objects it creates (except for objects returned as results).
- Take advantage of tools such as valgrind which probe for improper use of dynamic memory.
- And code the program very carefully!



Run-Time Failure Examples

Some Possibilities for Java (or C#)

- ArithmeticException
- ArrayIndexOutOfBoundsException
- NullPointerException
- ClassCastException
- OutOfMemoryException
- ... *the complete list is very long!*



Bug Classifications

- An early and still a definitive list of bug types is due to Donald Knuth.
- He kept a logbook of 850 coding errors when developing T_EX.
- He classified the errors into 15 categories.
- The list here is a condensed version due to Adam Barr. (*Find The Bug*, Addison-Wesley, 2005.)

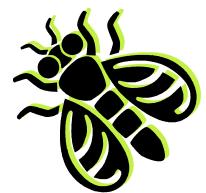
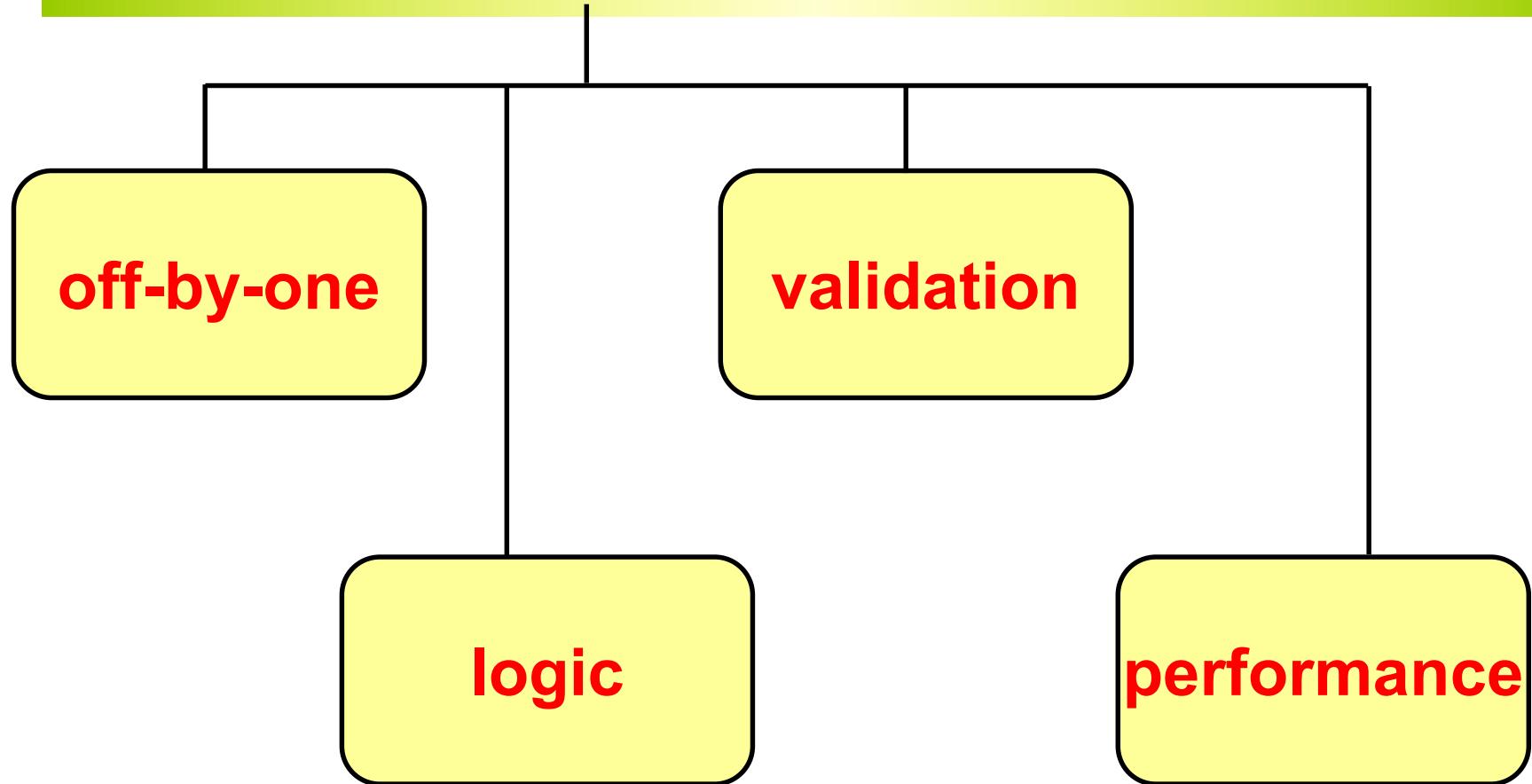


The Knuth/Barr Bug Categories

A – Algorithm	the code follows the intent of the programmer but the intent was wrong
D – Data	the code reads/writes incorrect data or accesses a wrong storage location
F – Forgotten	a missing action/statement or maybe it is in the wrong place in the program
B – Blunder	the algorithm was correct but it was implemented wrong or it was mistyped



A – Algorithm: Subcategories



A – Algorithm: Subcategories

off-by-one

Miscounting by one.

```
// count how many pages to print  
pageCount = lastPage - firstPage;  
  
// initialize the array  
for( int i=0; i<=A.length; i++ ) {  
    A[i] = 0;  
}
```



A – Algorithm: Subcategories

logic

The algorithm is simply wrong

```
// find the greatest difference  
// between any pair of elements  
biggest = 0.0;  
  
for( k = 0; k < A.length-1; k++ ) {  
    distance = abs(A[k]-A[k+1]);  
    if (distance>biggest)  
        biggest = distance;  
}
```



A – Algorithm: Subcategories

Failure to check
validity of variables

validation

```
// find average difference
float avgDiff( float A[], int n ) {
    int i; float sum = 0.0;
    for( i = 1; i<n; i++ )
        sum += abs(A[i]-A[i-1]);
    return sum/n;
}
```

But what happens if $n == 1$?



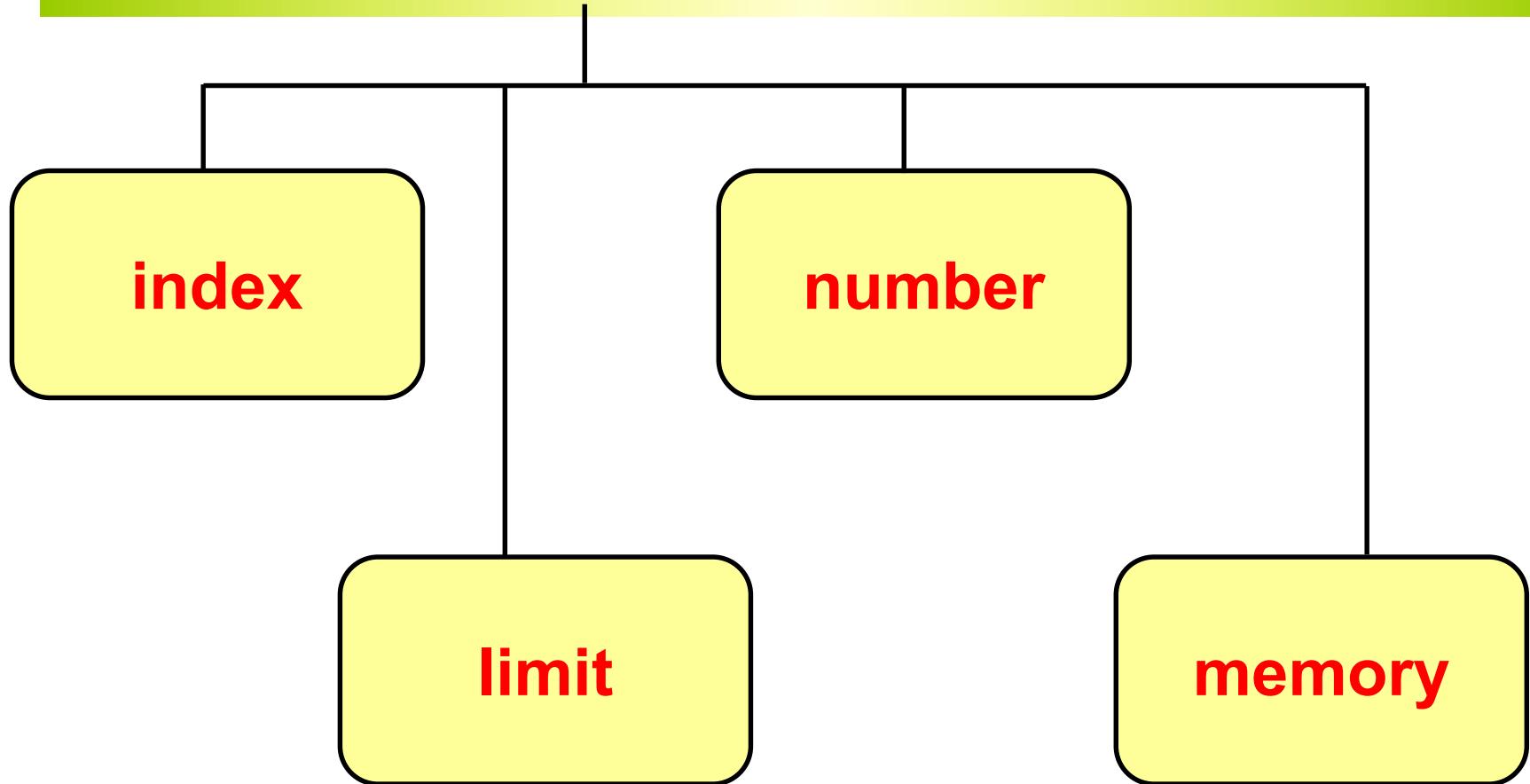
A – Algorithm: Subcategories

- an inappropriate algorithm, or
- a coding error which leads to unacceptable performance

performance



D – Data: Subcategories



D – Data: Subcategories

index

An array index is wrong.

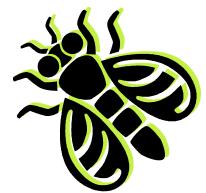
```
int search( int[] Array, int Val ) {  
    int Hi = Array.length; int Lo = 0;  
    while(Lo<=Hi) {  
        int Mid = (Hi+Lo)/2;  
        if (Array[Mid]==Val) return Mid;  
        if (Array[Mid]<Val)  
            Lo = Mid+1;  
        else  
            Hi = Mid-1;  
    }  
    return -1;  
}
```



D – Data: Subcategories

Incorrect handling for the first or last elements in a sequence.

limit



D – Data: Subcategories

Issues related to how numbers are represented in the computer

number

```
for( ; ; ) {  
    unsigned char c = getchar();  
    if (c == EOF) break;  
    ...  
}
```



D – Data: Subcategories

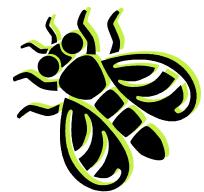
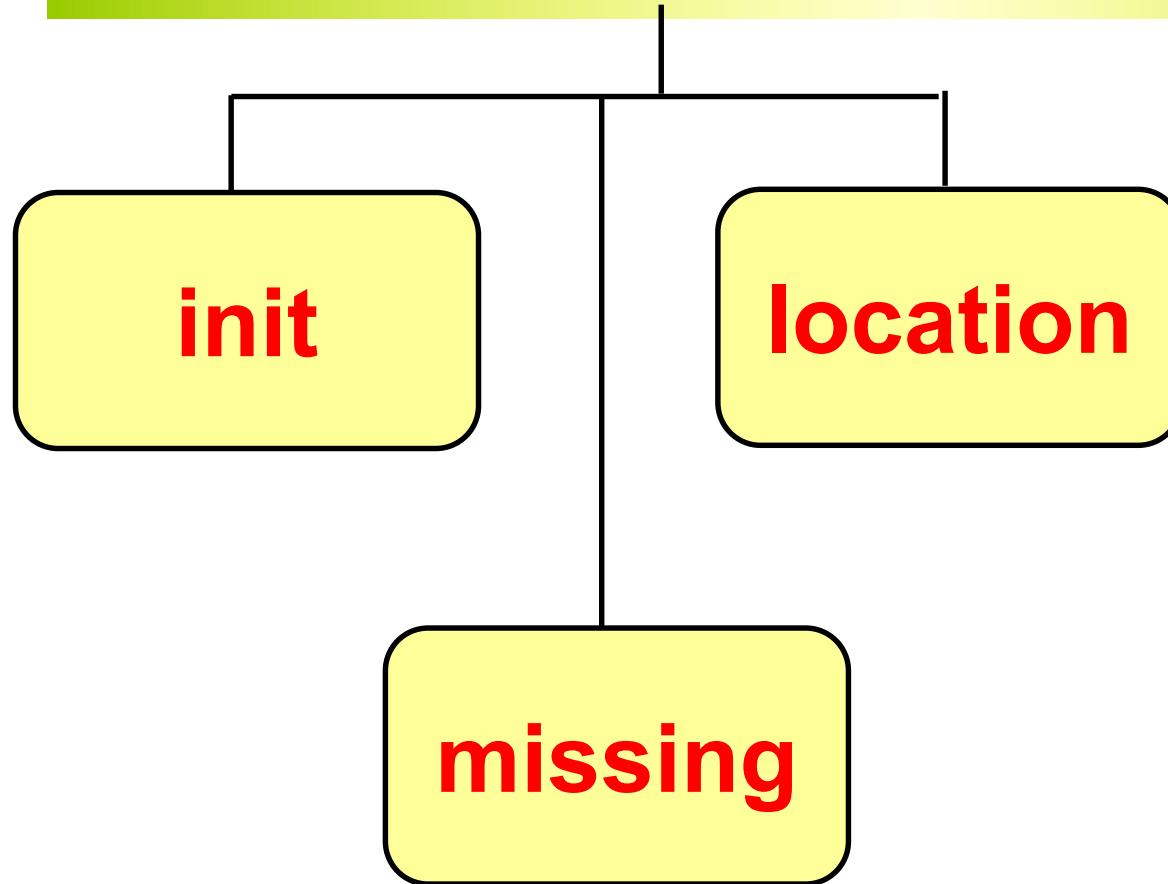
Mismanagement of memory – such as

- attempting to access an inaccessible location,
- or deallocated the same block twice, or
- failing to deallocate a block, or
- continuing to use a block after it has been deallocated.

memory



F – Forgotten: Subcategories



F – Forgotten: Subcategories

init

Forgetting to initialize a variable or a data structure

```
int biggest;  
int i;  
for( i=0; i<N; i++ ) {  
    if (A[i] > biggest) biggest = A[i];  
}
```



F – Forgotten: Subcategories

Forgetting to provide code
for some case.

missing

```
for( cnt=0; cnt<N; cnt++ ) {  
    val = ptr->v;  
    if (val < 0) continue;  
    sum += val;  
    ptr = ptr->next;  
}
```



F – Forgotten: Subcategories

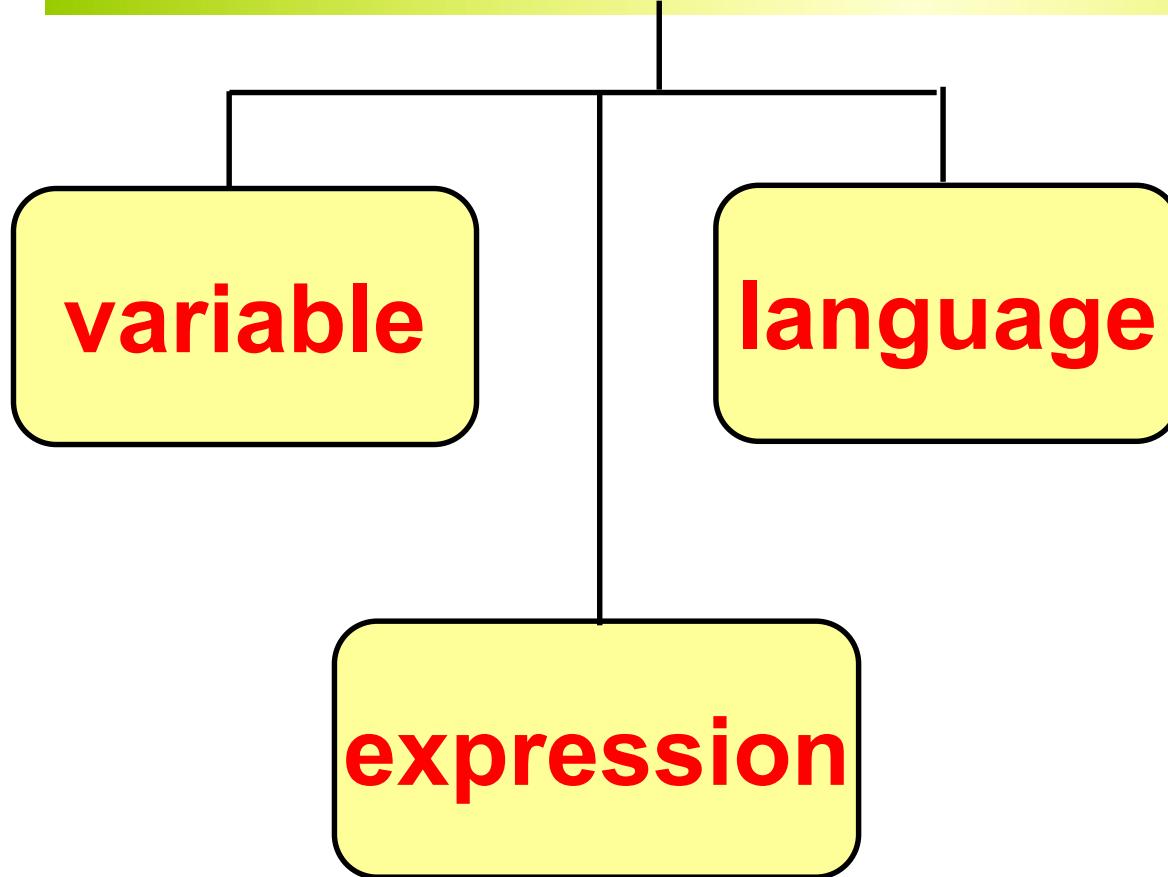
location

Code appears in the wrong place or is out of order.

```
array[ix] = 0;  
sum += array[ix];
```



B – Blunder: Subcategories



B – Blunder: Subcategories

variable

Simply writing the wrong
variable name

```
x1 = transform(x1, x2, delta_x);  
y1 = transform(y1, x2, delta_y);
```



Slide 36



B – Blunder: Subcategories

expression

An incorrectly written expression

```
if ((count < min) &&  
     (count > max)) ...
```



B – Blunder: Subcategories

language

A mistake encouraged by
the language syntax

==

if (a ~~==~~ b) { . . . }



Some Bug Avoidance Strategies

Use a High-Level Language

and obtain all the advantages of

- strong type-checking at compile time,
- memory management,
- run-time checking of pointers, array indexes, arithmetic overflow, etc. (usually!).

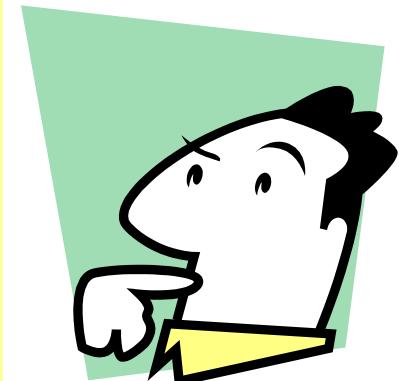


Some Bug Avoidance Strategies

Think Before You Code

Depending on the size of the programming team and the size/scope of the project, you may want to:

- Write out and debug the specifications
- Plan the software architecture
- Design the data structures
- Develop pseudocode for the algorithms
- Prove the correctness of the algorithms



Some Bug Avoidance Strategies

Write Robust Code – the strategies include:

validate

assertions

reuse

check results

KISS

watch out for gotchas



Some Bug Avoidance Strategies

Write Robust Code – the strategies include:

validate

assertions

reuse

check results

KISS

watch out for gotchas

Every subroutine should validate its inputs
(unless there is a clear *proof* that the inputs will always be valid)



Some Bug Avoidance Strategies

Write Robust Code – the strategies include:

validate

assertions

reuse

check results

KISS

watch out for gotchas

Insert assertions to verify that pointers are non-null and array indexes are in range (unless guaranteed by the programming language)



Some Bug Avoidance Strategies

Write Robust Code – the strategies include:

validate

assertions

reuse

check results

KISS

watch out for gotchas

Use library functions and library classes wherever possible (they have already been debugged!)



Some Bug Avoidance Strategies

Write Robust Code – the strategies include:

validate

assertions

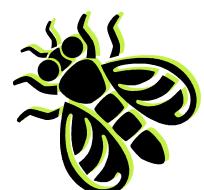
reuse

check results

KISS

watch out for gotchas

Library functions often return codes to indicate exceptional conditions – be sure to check them!



Some Bug Avoidance Strategies

Write Robust Code – the strategies include:

validate

assertions

reuse

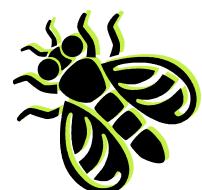
check results

KISS

watch out for gotchas

“Keep It Simple, Stupid!”

Avoid complicated logic in your programs; it will probably cost you more debugging time than it saves in execution time



Some Bug Avoidance Strategies

Write Robust Code – the strategies include:

validate

assertions

reuse

check results

KISS

watch out for gotchas

Many programming languages have common pitfalls, such as

`if (a = true) { ... }`

in C, C++, Java, C#.



Some Bug Avoidance Strategies

Test Your Code!

- Testing actually *finds* bugs rather than avoids bugs, but a failure which is found early (before releasing the program) and on a simple test case is much easier to analyze.
- Including test methods in classes (or test functions in modules) is a good idea.

