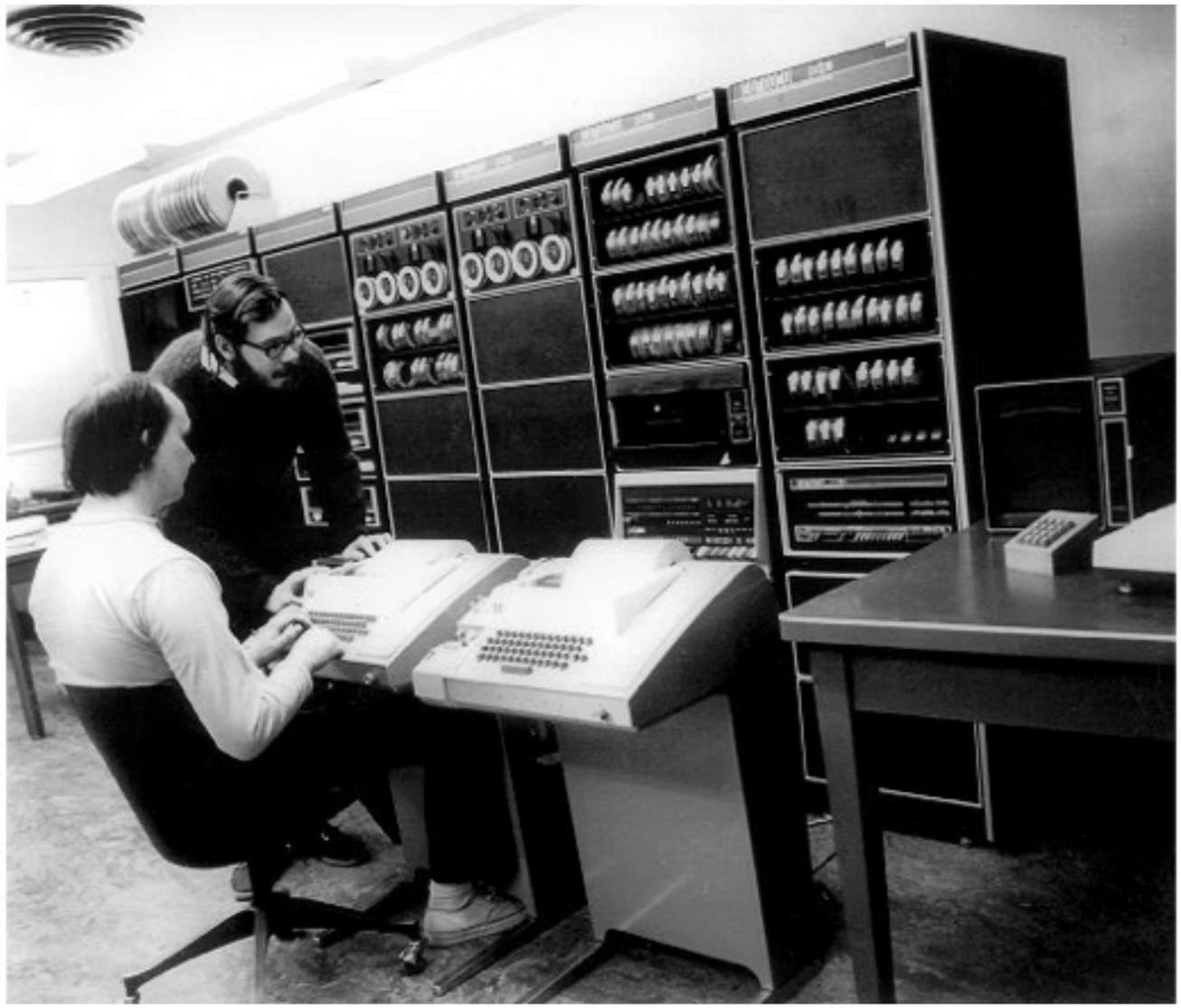
Control Flow

"to get a deeper understanding of the language"



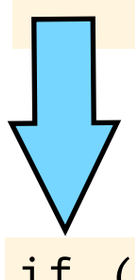
Deep C - a 3 day course
Jon Jagger & Olve Maudal

dangling else

- in a nested if an else associates with its nearest lexical if

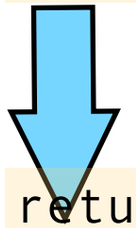
```
if (value >= 0)
    if (value <= 100)
        return true;
else
    return false;
```

physical indentation != logical indentation



```
if (value >= 0)
    if (value <= 100)
        return true;
    else
        return false;
```


physical indentation == logical indentation



```
return (value >= 0) && (value <= 100);
```

```
return (0 <= value) && (value <= 100);
```

discussion

- using = instead of == is a common bug
- compiles because assignment is an expression

```
if (x = 42)
    ...
```

oops: but not a
compile time error

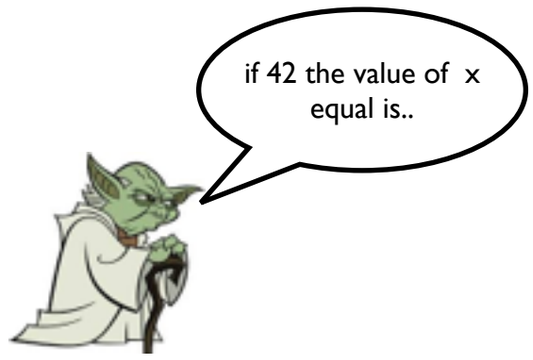
```
if (x == 42)
    ...
```

- how about reversing the operands?
- a common guideline, but what do you think?

```
if (42 = x)
    ...
```

compile time error

```
if (42 == x)
    ...
```



```
if (42 == x)
    ...
```

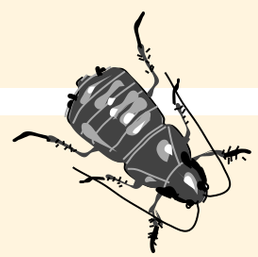
My advice is that writing if (42 == x) is not a good idea. It makes the code harder to read. If you are aware enough of the problem to write the variable on the right hand side surely you are aware enough of the problem to write == instead of =? And it doesn't apply all the time. What about of both arguments are variables? Most crucially of all, writing if (x = 42) should be found by tests. Time and time again studies have shown the two major factors in building software are (1) how interdependent the various parts of your software are – so that when you change one part, only some of the rest is affected, and (2) how easy the code is to comprehend.

When programming, **readability** should have very high priority!

compound literals c99

- compound literals inside a function have ***automatic*** storage duration
- their lifetime is the enclosing block
- an if has a block even without { braces }

```
int * danger(int n)
{
    if (n % 2 == 0)
        return (int[]){ 1,2,3,4,5 };
    else
        return (int[]){ 5,4,3,2,1 };
}
```



compiler rewrites

```
int * danger(int n)
{
    if (n % 2 == 0)
    {
        int _t[] = { 1,2,3,4,5 };
        return _t;
    }
    else
    {
        int _t[] = { 5,4,3,2,1 };
        return _t;
    }
}
```

no gcc warning

fall through

- case labels provide "goto" style jump points
- when inside the switch they play no part
- there is no implicit break
- this is known as fall-through

```
void send(short * to, short * from, int count)
{
    int n = (count + 7) / 8;
    switch (count % 8)
    {
    case 0 : do { *to++ = *from++;
    case 7 :      *to++ = *from++;
    case 6 :      *to++ = *from++;
    case 5 :      *to++ = *from++;
    case 4 :      *to++ = *from++;
    case 3 :      *to++ = *from++;
    case 2 :      *to++ = *from++;
    case 1 :      *to++ = *from++;
                } while (--n > 0);
    }
}
```

Duff's Device

switch gotchas

will this assignment happen?

```
int at;
...
switch (days % 10)
{
    at = 0;

case 1 :
    while (at != 0) ... break;
case 2 :
    while (at != 0) ... break;
case 3 :
    while (at != 0) ...
defualt:
    error(); break;
}
```

how do you spell default?

fall-through is usually an error

why write this break?

for

- declared variables are scoped to the *for* statement c99

```
enum { max_size = 42; };
int array[max_size] = { ... };
```

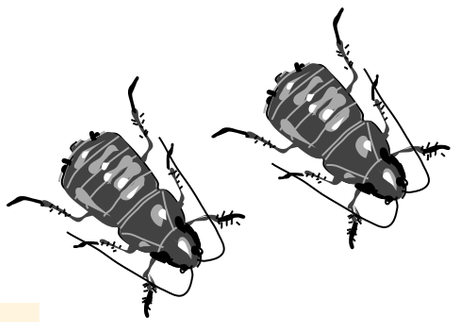
```
for (size_t i = 0; i <= max_size; i++) {
    foo(array[i]);
}
i = 0;
```

```
for (size_t i = 0; i <= max_size; i++)
    foo(array[i]);

i = 0;
```

continue

- starts a new iteration of nearest enclosing while/do/for
- highly correlated with bugs



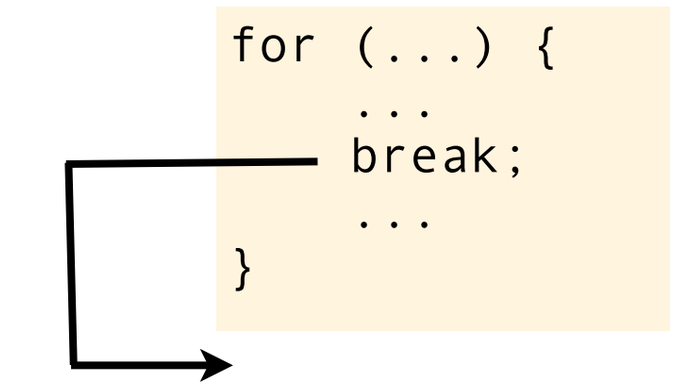
```
for (int at = 0; at != 0; at++) {
    ...
    continue;
    ...
}
```

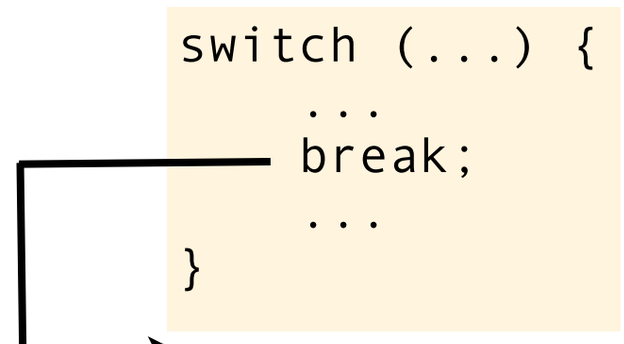
```
do {
    ...
    continue;
    ...
}
while (...);
```

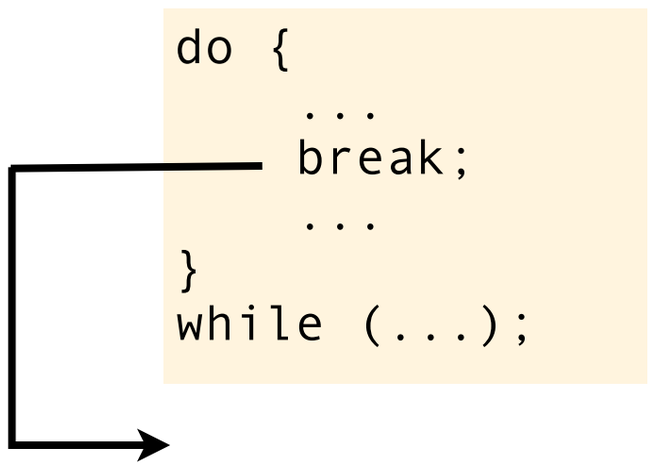
```
while (...) {
    ...
    continue;
    ...
}
```

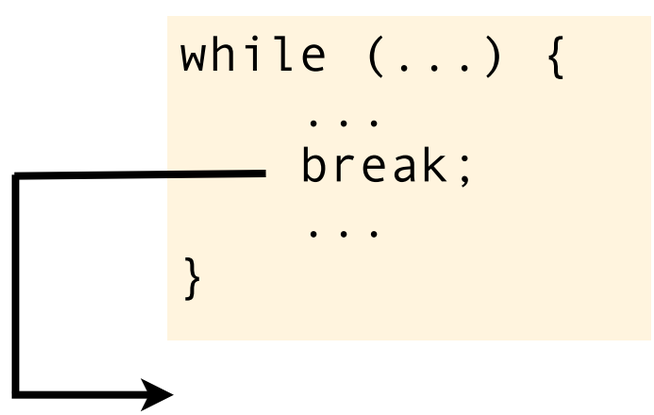
break

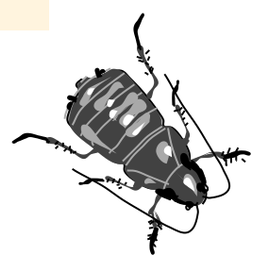
- exits nearest enclosing while/do/for/switch
- correlated with bugs in loops
- normal in a switch statement











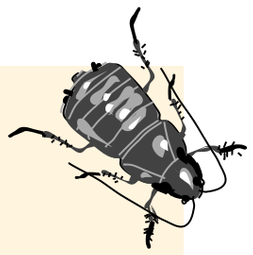
6.2.4 Storage duration of objects

An object has a storage duration that determines its lifetime.
There are four storage durations: static, thread, ***automatic***, and allocated.
...
An object whose identifier is declared with no linkage and without the storage class specifier static has automatic storage duration...
...
The initial value of the object is ***indeterminate***.

6.3.2.1 Lvalues ...

An lvalue designates an expression...
If the lvalue designates an object of ***automatic*** storage duration... and that object is uninitialised... the behaviour is ***undefined***.

```
int main(void)
{
    int n;
    int v = n;
    ...
}
```

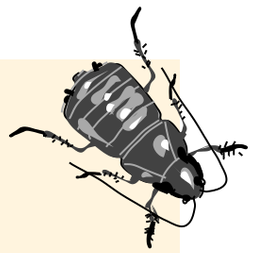


undefined

goto

- jumps to the labelled statement, not the label
- can jump forward or backward
- can bypass variable initialization!

```
void some_function(void)
{
    goto bypass;
    int n = 42;
bypass:
    printf("%d\n", n);
}
```



undefined

-Werror=uninitialized

return

- if you use the returned expression, the return must return something!

```
const char * day_suffix(int day)
{
    if (day / 10 == 1)
        return "th";
    else
        switch (day % 10)
        {
        case 1: return "st";
        case 2: return "nd";
        case 3: return "rd";
        }
}
```

-Werror=return-type (control reaches end of non-void function)

```
int main(void)
{
    assert(strcmp(day_suffix(13),"th") == 0);
    assert(strcmp(day_suffix( 5),"th") == 0);
}
```

undefined

_Noreturn c11

- _Noreturn inicates the function does not return
- #include <stdnoreturn.h> provides a noreturn macro

<stdlib.h>

```
_Noreturn void abort(void);
_Noreturn void exit(int status);
_Noreturn void _Exit(int status);
_Noreturn void quick_exit(int status);
```

```c
#include <stdlib.h>
#include <stdnoreturn.h>

noreturn void fatal_error(int status)
{
    fprintf(stderr, "fatal error");
    exit(status);
    // any code here cannot be reached
}
```

-Werror=missing-noreturn

summary

- if : beware of dangling else
- if : avoid yoda-speak
- if : be careful with compound-literals scope
- switch : beware fall-though
- for : iteration variable is scoped to the for loop
- continue, break, goto : all correlated with bugs
- noreturn : functions that don't return (c11)