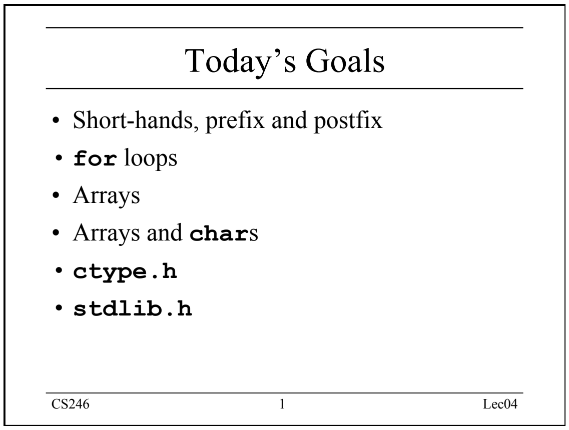
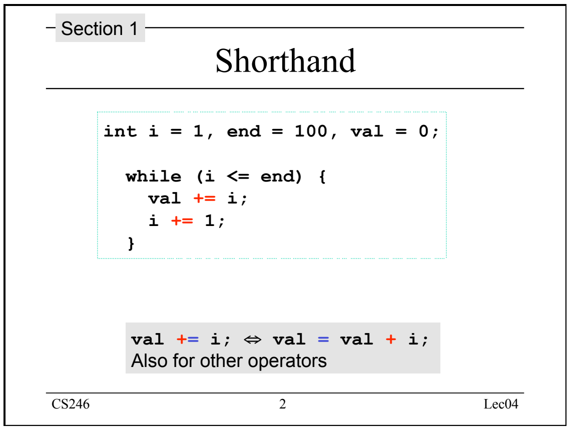


Today’s Goals

- Short-hands, prefix and postfix
- `for` loops
- Arrays
- Arrays and `char`s
- `ctype.h`
- `stdlib.h`

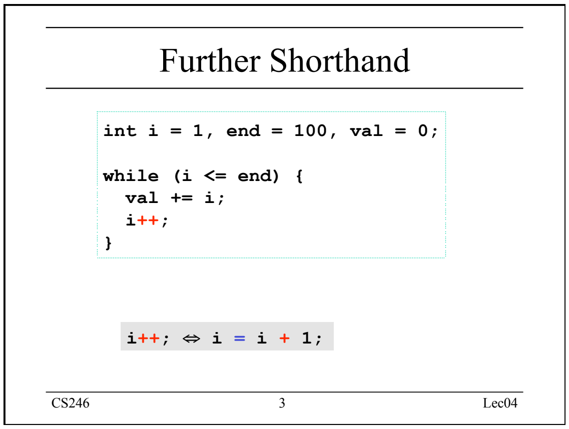


Section 1

Shorthand

```
int i = 1, end = 100, val = 0;

  while (i <= end) {
    val += i;
    i += 1;
  }
```

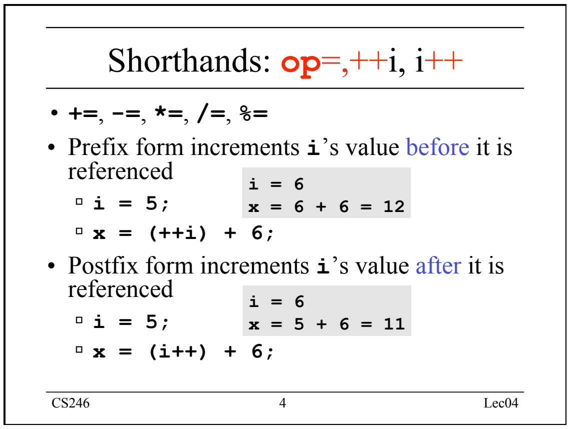
`val += i;` ⇔ `val = val + i;`
Also for other operators



Further Shorthand

```
int i = 1, end = 100, val = 0;

while (i <= end) {
  val += i;
  i++;
}
```

`i++;` ⇔ `i = i + 1;`



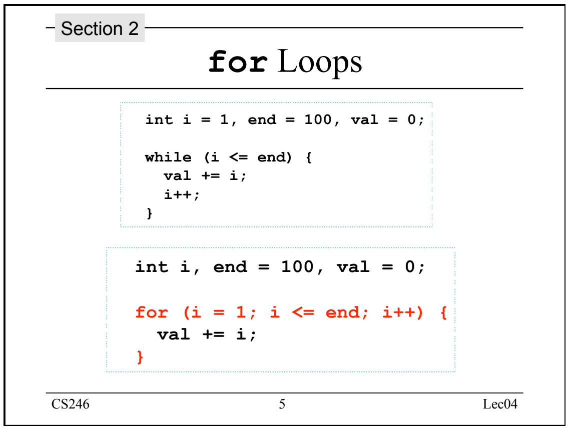
Shorthands: op=,++i, i++

- `+=`, `-=`, `*=`, `/=`, `%=`
- Prefix form increments `i`’s value before it is referenced
  - `i = 5;`
  - `x = (++i) + 6;`

```
i = 6
x = 6 + 6 = 12
```

- Postfix form increments `i`’s value after it is referenced
  - `i = 5;`
  - `x = (i++) + 6;`

```
i = 6
x = 5 + 6 = 11
```

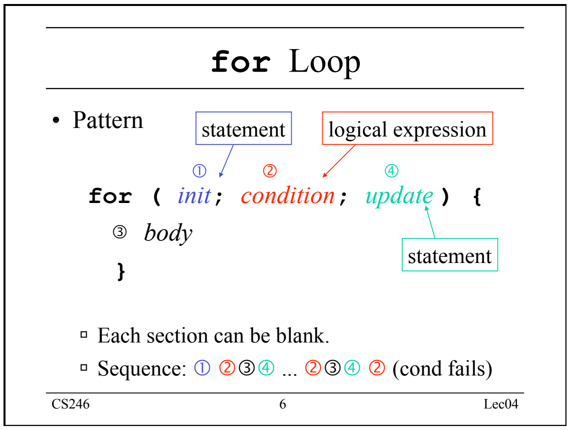


Section 2

for Loops

```
int i = 1, end = 100, val = 0;

while (i <= end) {
  val += i;
  i++;
}
```

```
int i, end = 100, val = 0;

for (i = 1; i <= end; i++) {
  val += i;
}
```



for Loop

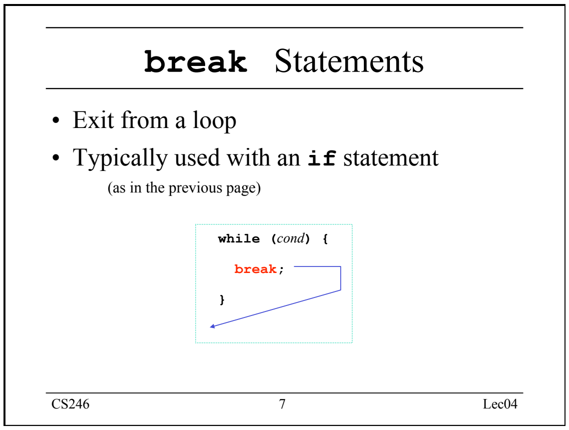
- Pattern

statement   logical expression

```
for ( ①init; ②condition; ④update ) {
  ③ body
}
```

statement

- Each section can be blank.
- Sequence: ① ②③④ ... ②③④ ② (cond fails)

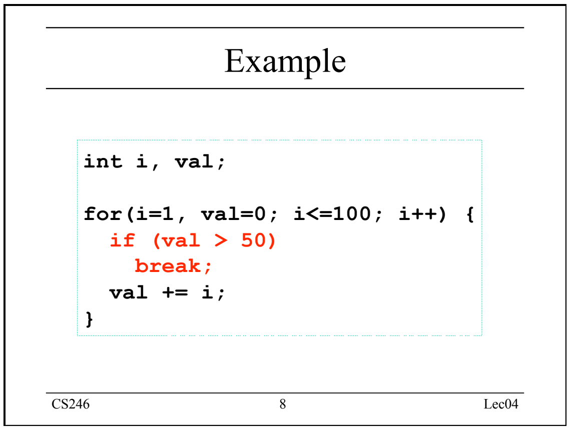


break Statements

- Exit from a loop
- Typically used with an **if** statement

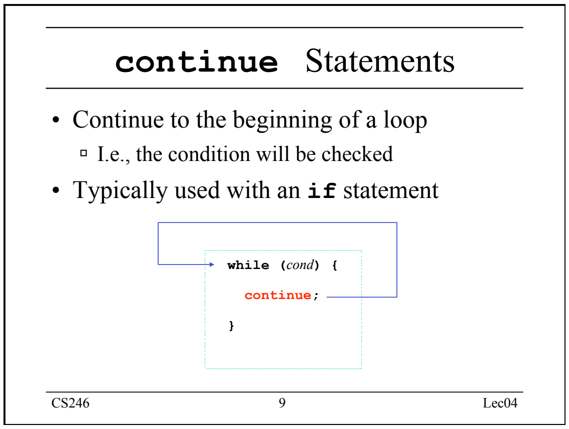
(as in the previous page)

```
while (cond) {
   break;
}
```



Example

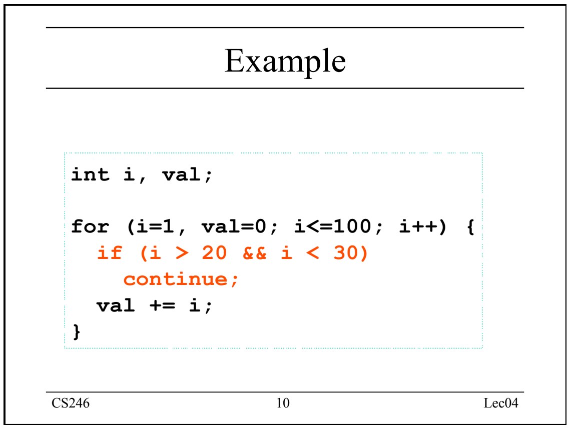
```
int i, val;

for(i=1, val=0; i<=100; i++) {
  if (val > 50)
    break;
  val += i;
}
```



continue Statements

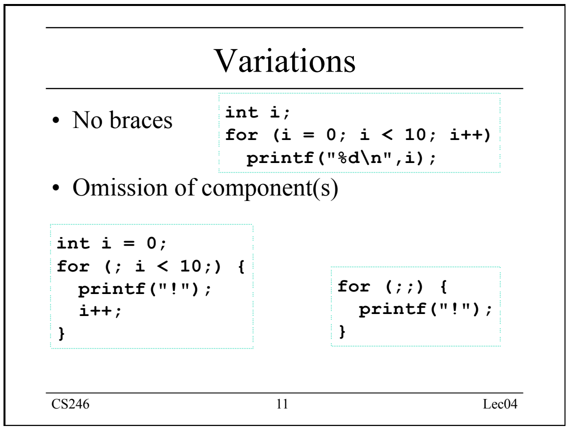
- Continue to the beginning of a loop
  - I.e., the condition will be checked
- Typically used with an **if** statement

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



Example

```
int i, val;

for (i=1, val=0; i<=100; i++) {
  if (i > 20 && i < 30)
    continue;
  val += i;
}
```



Variations

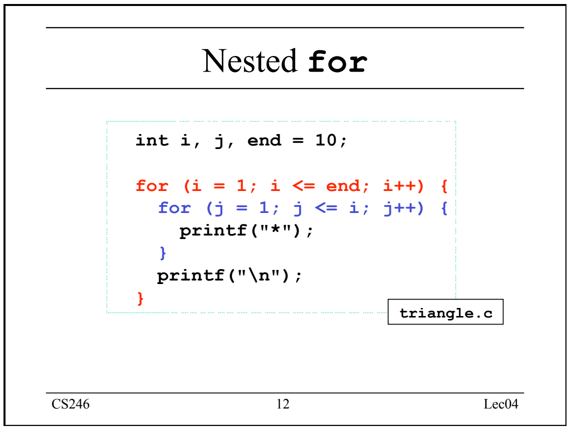
- No braces

```
int i;
for (i = 0; i < 10; i++)
  printf("%d\n",i);
```

- Omission of component(s)

```
int i = 0;
for (; i < 10;) {
  printf("!");
  i++;
}
```

```
for (;;) {
  printf("!");
}
```



Nested **for**

```
int i, j, end = 10;

for (i = 1; i <= end; i++) {
  for (j = 1; j <= i; j++) {
    printf("*");
  }
  printf("\n");
}
```

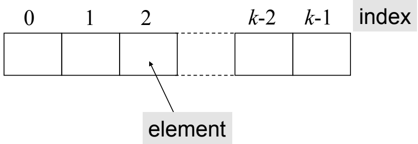
triangle.c

Section 3

Arrays

Flippo Brunelleschi
Ospdale degli Innocenti
Firenze, Italy, 1421

- To store a large number of data of homogenous type (e.g. `int` only)
- Schematic representation



Array Operations

- Declaration

  `int a[5];`

  size

- Assignment

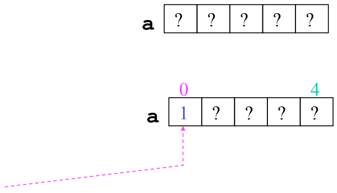
  `a[0] = 1;`

  index

- Reference

  `int y = a[0];`

  index



Arrays and Characters

```
int main() {
  int digits[10] = {0}, i; char c;

  while((c = getchar()) != EOF) {
    if (c >= '0' && c <= '9')
      digits[c-'0']++;
  }

  return 0;
}
```

digitcount.c

Section 4

ctype.h

- C library containing a bunch of very useful character functions.
- These functions take an integer (not necessarily a **char**!) and return **0** or **1**.
- **int isdigit(int c);**
- **isalpha, isalnum, isspace, islower, isupper**
- **int tolower/toupper (int c);**

stdlib.h

- **void exit(int status);**
- Terminates a C program.
- Non-zero parameter values indicate program error to parent.
- A call to **exit(1)** is often used in conjunction with error detection.

Summary

- Be careful with prefix and postfix, especially postfix
- Loops are where a program spends most of its time. Learn to write efficient ones!
- Learn to use array and characters flexibly