Section 8: Loops

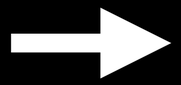
```
while 1 do
 puts "hello, world!"
end
```

Samuel Wolfson // CSE 120 18wi

`while` Loops

```
while (condition) {
 // statements
}
```

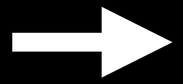
`while` Loop Example



```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```

x	
prod	
x > 1	

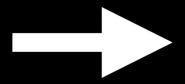
`while` Loop Example



```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```

x	4
prod	
x > 1	

`while` Loop Example

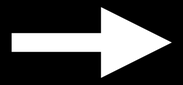


```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```

x	4
prod	1
x > 1	true

`while` Loop Example

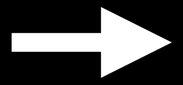
```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```



x	4
prod	1
x > 1	true

`while` Loop Example

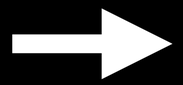
```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```



x	4
prod	4
x > 1	true

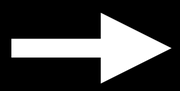
`while` Loop Example

```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```



x	3
prod	4
x > 1	true

`while` Loop Example

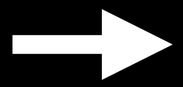


```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```

x	3
prod	4
x > 1	true

`while` Loop Example

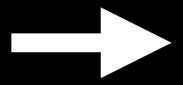
```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```



x	3
prod	4
x > 1	true

`while` Loop Example

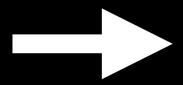
```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```



x	3
prod	12
x > 1	true

`while` Loop Example

```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```



x	2
prod	12
x > 1	true

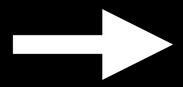
`while` Loop Example

```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```

x	2
prod	12
x > 1	true

`while` Loop Example

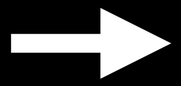
```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```



x	2
prod	12
x > 1	true

`while` Loop Example

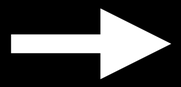
```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```



x	2
prod	24
x > 1	true

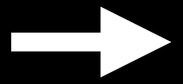
`while` Loop Example

```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```



x	1
prod	24
x > 1	true

while Loop Example



```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```

x	1
prod	24
x > 1	false

`while` Loop Example

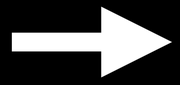
```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```

x	1
prod	24
x > 1	false

`while` Loop Example

```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}

println("The answer is: " + prod);
```

x	1
prod	24
x > 1	false



```
> The answer is: 24
```

`for` loops

How can we translate our previous example into a `for` loop?

```
for (initialization; condition; increment) {
 // statements
}
```

`for` loops

How can we translate our previous example into a `for` loop?

```
for (initialization; condition; increment) { /* statements */ }
```

```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}
```

```
int x = 4;
int prod = 1;

for (    ;     ;    ) {
    prod = prod * x;
    x = x - 1;
}
```

```
println("The answer is: " + prod);
```

`for` loops

How can we translate our previous example into a `for` loop?

```
for (initialization; condition; increment) { /* statements */ }
```

```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}
```

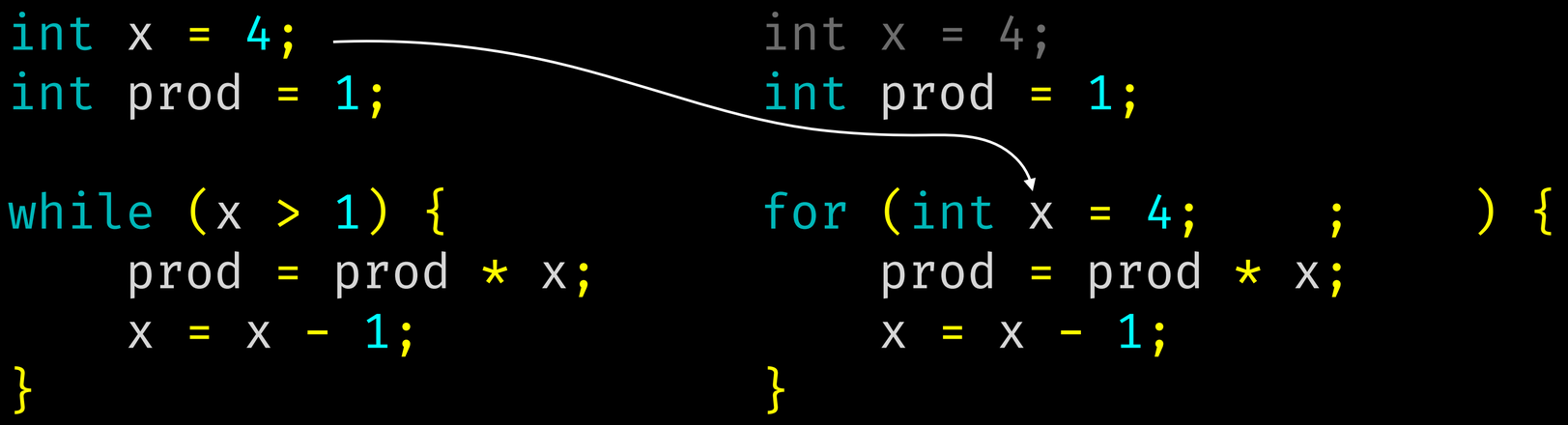
```
int x = 4;
int prod = 1;

for (    ;     ;    ) {
    prod = prod * x;
    x = x - 1;
}
```

```
println("The answer is: " + prod);
```

`for` loops

How can we translate our previous example into a `for` loop?

```
for (initialization; condition; increment) { /* statements */ }
```



```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}
```

```
int x = 4;
int prod = 1;

for (int x = 4;     ;     ) {
    prod = prod * x;
    x = x - 1;
}
```

```
println("The answer is: " + prod);
```

`for` loops

How can we translate our previous example into a `for` loop?

```
for (initialization; condition; increment) { /* statements */ }
```

```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}
```

```
int x = 4;
int prod = 1;

for (int x = 4;     ;     ) {
    prod = prod * x;
    x = x - 1;
}
```

```
println("The answer is: " + prod);
```

`for` loops

How can we translate our previous example into a `for` loop?

```
for (initialization; condition; increment) { /* statements */ }
```

```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}
```

```
int x = 4;
int prod = 1;

for (int x = 4; x > 1;    ) {
    prod = prod * x;
    x = x - 1;
}
```

```
println("The answer is: " + prod);
```

`for` loops

How can we translate our previous example into a `for` loop?

```
for (initialization; condition; increment) { /* statements */ }
```

```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}
```

```
int x = 4;
int prod = 1;

for (int x = 4; x > 1;    ) {
    prod = prod * x;
    x = x - 1;
}
```

```
println("The answer is: " + prod);
```

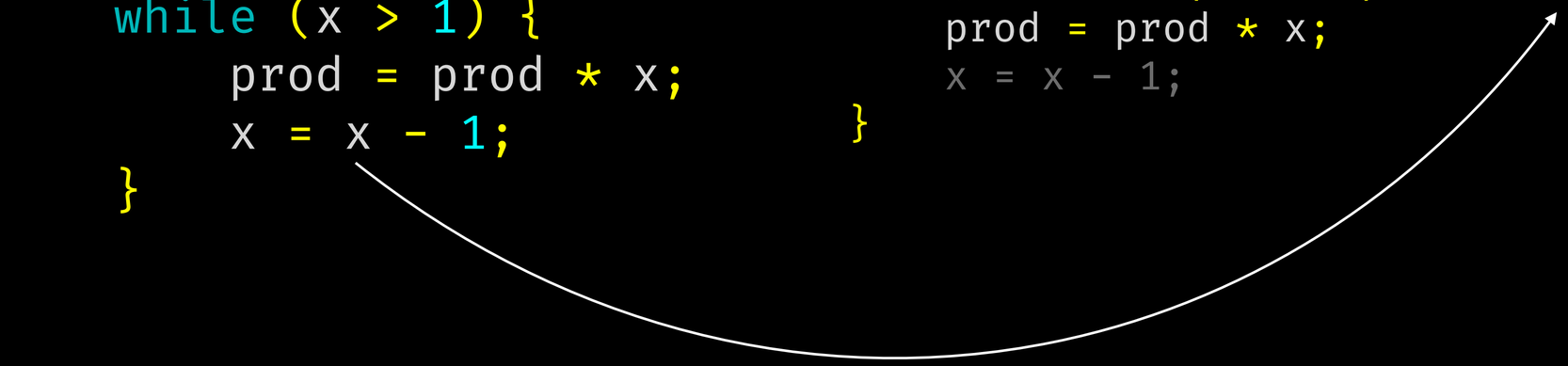
`for` loops

How can we translate our previous example into a `for` loop?

```
for (initialization; condition; increment) { /* statements */ }
```

```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}
```

```
int x = 4;
int prod = 1;

for (int x = 4; x > 1; x = x - 1) {
    prod = prod * x;
    x = x - 1;
}
```



```
println("The answer is: " + prod);
```

`for` loops

How can we translate our previous example into a `for` loop?

```
for (initialization; condition; increment) { /* statements */ }
```

```
int x = 4;
int prod = 1;

while (x > 1) {
    prod = prod * x;
    x = x - 1;
}
```

```
int x = 4;
int prod = 1;

for (int x = 4; x > 1; x = x - 1) {
    prod = prod * x;
    x = x - 1;
}
```

```
println("The answer is: " + prod);
```

`for` loops

How can we translate our previous example into a `for` loop?

```
for (initialization; condition; increment) { /* statements */ }
```

```
int x = 4;
int prod = 1;

for (int x = 4; x > 1; x = x - 1) {
    prod = prod * x;
    x = x - 1;
}
```

```
println("The answer is: " + prod);
```

`for` loops

How can we translate our previous example into a `for` loop?

```
for (initialization; condition; increment) { /* statements */ }
```

```
int prod = 1;

for (int x = 4; x > 1; x = x - 1) {
    prod = prod * x;
}
```

```
println("The answer is: " + prod);
```

`for` loops

How can we translate our previous example into a `for` loop?

```
for (initialization; condition; increment) { /* statements */ }
```

```
int prod = 1;

for (int x = 4; x > 1; x = x - 1) {
    prod = prod * x;
}

println("The answer is: " + prod);
```

`for` loops

How can we translate our previous example into a `for` loop?

```
for (initialization; condition; increment) { /* statements */ }
```

```
int prod = 1;

for (int x = 4; x > 1; x--) {
    prod = prod * x;
}

println("The answer is: " + prod);
```

Side note: `x--` is shorthand for `x = x - 1`

Other Fun Loops (C)

```
for (char **ptr = argv; *ptr != 0; ptr++) {
    char *str;
    if (fetchstr((uint64_t) *ptr, &str) < 0) {
        return -1;
    }
}
```

Other Fun Loops (C)

```c
while (index < MAXARG && argv[index] =/= NULL) {
    // how much space do we need for this string (with \0)?
    int len = strlen(argv[index]) + 1;

    // decrement stack pointer
    sp -= len;

    // write the string to the new vspace
    vspacewritetova(&newSpace, sp, argv[index], len);

    // record where we put it
    arg_ptrs[index] = (char *) sp;

    index++;
}
```

Other Fun Loops (C++)

```
std::unordered_map<string, HWSize_t> retval;

for (auto const &it : runningMatches) {
    std::string docname;
    dtr->LookupDocID(it.first, &docname);
    retval[docname] = it.second;
}
```

Other Fun Loops (Python)

```
for group in assn_groups:
        assignments = group.get_assignments()
        for assignment in assignments:
            if 'checkoff' in assignment.name.lower():
                a_model = Assignment(canvas_id=assignment.id,
                                     name=assignment.name,
                                     points=assignment.points,
                                     due_at=assignment.due_at)

                a_model.save()
```

Other Fun Loops (Embedded Ruby)

```
<% @users.each do |user| %>
      <tr>
        <td><%= user.name %></td>
        <td><%= user.lizard_name %></td>
        <td><%= link_to 'Show', user %></td>
        <td><%= link_to 'Edit', edit_user_path(user) %></td>
        <td><%= link_to 'Destroy', user, method: :delete, data:
{ confirm: 'Are you sure?' } %></td>
      </tr>
<% end %>
```