CSCI 135
Control Flow (Iteration)

Subash Shankar

Iteration

So far:

- Assignment: update state (*i.e.*, value of variable)
- Control (selection): evaluate a statement, and execute different statement depending on the result of the evaluation

We still need a way to repeatedly execute a statement.
$\Rightarrow$ Iteration.
Iteration Constructs in C/C++:

- `while:` execute a statement as long as some condition is true
- `do while:` similar to `while` but check condition *after* executing statement.
- `for:` execute a statement on every element of some data, or some fixed number of times.

⚠Above uses are typical, but there are always exceptions (and they all have the same expressive power).

Iteration: While

while <cond> S

where S (called **loop body**) is a [simple or compound] statement, and cond is a boolean expression as in selection)
Execute S as long as cond is true, where cond is evaluated before entering loop body (each time).

Ex:

```
int count = 0;
while (count < 3) {
  cout << "Hello " << count << endl;
  count ++;
}
```

prints `Hello 0`, `Hello 1`, `Hello 2` (on 3 lines).

Example: Add 1 to inputs

Spec: Input a sequence of numbers, $\{a_i\}$, and output the sequence $\{a_i+1\}$. Terminate when the input is -1.

```
while (a != -1) {
   cin >> a;
   cout << (a+1) << endl;
}
```

? Any bugs?

Example: Add 1 to inputs

Spec: Input a sequence of numbers, $\{a_i\}$, and output the sequence $\{a_i+1\}$. Terminate when the input is -1.

```
while (a != -1) {
   cin >> a;
   cout << (a+1) << endl;
}
```

? Any bugs?

! It also prints an output for the final input (-1+1)

⇒ Need to `cin` before the loop starts

Add 1 to inputs - Attempt 2

Spec: Input a sequence of numbers, $\{a_i\}$, and output $\{a_i + 1\}$. Terminate when the input is -1.

```
cin >> a;                          input a before loop
while (a != -1) {
   cout << (a+1) << endl;
   cin >> a;                       input a for next iteration
}
```

This works (though code is a more confusing).

? What was the real problem?

Add 1 to inputs - Attempt 2

Spec: Input a sequence of numbers, $\{a_i\}$, and output $\{a_i + 1\}$. Terminate when the input is -1.

```
cin >> a;                          input a before loop
while (a != -1) {
  cout << (a+1) << endl;
  cin >> a;                        input a for next iteration
}
```

This works (though code is a more confusing).

? What was the real problem?

! We needed to terminate the loop based on state after `cin` ($\equiv$ end of loop body), not beginning of loop body.

Add 1 to inputs - Attempt 3

while <cond> S

Execute S as long as cond is true, evaluating cond *before* loop entry.

do S while <cond>

Similar to `while` but evaluate cond at *end* of loop body S. (thus, S is always executed at least once)

```
do {
    cin >> a;
    if (a != -1) cout << (a+1) << endl;
} while (a != -1)
```

a is still the input value now

Cleaner and more readable than attempt 2 (maybe)!

Example: Print *num* asterisks on one line

```
cin >> num;
int ctr = 0;          need counter to keep track
while (ctr<num) {
  cout << '*';
  ctr++;              move (towards termination condition)
};
cout << endl;         don't forget the new line!
```

This works, but ctr update is control logic and extraneous to main point of loop body (⇒ harder to understand, maintain)

Print *num* asterisks on one line - Attempt 2

for (<init_action>; <cond>; update_action) S

1. Perform init_action (to establish initial state)
2. Check cond; exit loop if false
3. Execute S
4. Perform update_action
5. Goto step 2

```
for (int ctr=0; ctr<num; ctr++)

  cout << '*';
cout << endl;
```

Print *num* asterisks on one line - Attempt 2

for (<init_action>; <cond>; update_action) S

1. Perform init_action (to establish initial state)
2. Check cond; exit loop if false
3. Execute S
4. Perform update_action
5. Goto step 2

```
for (int ctr=0; ctr<num; ctr++)
```

↑ ctr is not needed/visible outside block

```
  cout << '*';
cout << endl;
```

Print *num* asterisks on one line - Attempt 2

for (<init_action>; <cond>; update_action) S

1. Perform init_action (to establish initial state)
2. Check cond; exit loop if false
3. Execute S
4. Perform update_action
5. Goto step 2

```
for (int ctr=0; ctr<num; ctr++)
```

↑ ctr is not needed/visible outside block

```
  cout << '*';
cout << endl;
```

Typical uses:

- Do something n times, with a counter variable
- Do something for every piece of [linearly structured] data

Add 1 to inputs (using `for`)

Code using while:

```
cin >> a;                          initialization action
while (a != -1) {                  loop condition
  cout << (a+1) << endl;
  cin >> a;
};                                 no update action here
```

Code using for:

```
for (cin>>a; a != -1;) {           null update action
  cout << (a+1) << endl;
  cin >> a;
};
```

This works, though do-while (from earlier) is the better (*i.e.*, more readable) version here.

Some Common Loop Errors

- ⚠ `while(<cond>); S`
  The semicolon terminates the loop! ⇒ if `cond` evaluates to true, its an infinite loop and `S` is never executed; Otherwise, S is executed exactly once (after loop).
- ⚠ `while(<cond>) S` where `S` does not update any variable in `cond` (or `cond` has no variables).
  ⇒ `cond` evaluates to the same value on every iteration, resulting in an infinite (or never-executed) loop
- ⚠ `while(n<64) S` where S doesn't increase `n`'s value.
  ⇒ Loop body needs to move *towards* making condition false.
- ⚠ Not initializing variables in `cond` before reaching loop.
- ⚠ 'Off by 1' errors:
  `for (i=0; i<=16; i++) S` would execute `S` 17 times (i=0,1,...,16).

Determining the Iteration Variable

? What is the iteration variable (and its range) if we want to compute $(n-a)+(n-a-1)+(n-a-2)+\ldots+(n-b)$ (given n,a,b)

Determining the Iteration Variable

? What is the iteration variable (and its range) if we want to compute $(n-a)+(n-a-1)+(n-a-2)+\ldots+(n-b)$ (given n,a,b)

Above is $\Sigma_{i=0}^{???}(n-a-i)$

Determining the Iteration Variable

? What is the iteration variable (and its range) if we want to compute $(n-a)+(n-a-1)+(n-a-2)+\ldots+(n-b)$ (given n,a,b)

Above is $\Sigma_{i=0}^{???}(n-a-i) = \Sigma_{i=0}^{b-a}(n-a-i)$

```
int s=0;
for (int i=0; i<=b-a; i++)
  s += n-a-i;
```

Determining the Iteration Variable

? What is the iteration variable (and its range) if we want to compute $(n-a)+(n-a-1)+(n-a-2)+\ldots+(n-b)$ (given n,a,b)

Above is $\Sigma_{i=0}^{???}(n-a-i) = \Sigma_{i=0}^{b-a}(n-a-i)$

```
int s=0;
for (int i=0; i<=b-a; i++)
  s += n-a-i;
```

? Easier way to do this?

Complex Conditions

Spec: Input a sequence of strings until two consecutive inputs are the same. Output the first character of each string. But, terminate if 32 strings have been input.

? When do we terminate loop?

Complex Conditions

Spec: Input a sequence of strings until two consecutive inputs are the same. Output the first character of each string. But, terminate if 32 strings have been input.

```
int num=0, max=32;
bool done = false;
string s="", s_old="";
while (!done) {
  cin >> s;
  done = ( done | (s == s_old) );
  s_old = s;
  if (s.length()>0) cout << s[0];
    else cout << '?';
  num++;  done = ( done | (num == max) );
};
cout << endl;
```

Better than hardwiring 32 in code!

incomplete spec!

Complex Conditions

Spec: Input a sequence of strings until two consecutive inputs are the same. Output the first character of each string. But, terminate if 32 strings have been input.

```
int num=0, max=32;
bool done = false;
string s="", s_old="";
while (!done) {
  cin >> s;
  done = ( done | (s == s_old) );
  s_old = s;
  if (s.length()>0) cout << s[0];
    else cout << '?';
  num++;  done = ( done | (num == max) );
};
cout << endl;
```

Better than hardwiring 32 in code!

incomplete spec!

What if first input is empty string?

Iteration Guidelines and Caveats

- Pick loop construct based on:
  - Do you know how many iterations, or is it based on a condition?
  - When do you want condition evaluated?
  - Now, which maps best onto problem (so that code is readable)?

  (though 3 types have equivalent expressivity)
- Use proper indentation of loop bodies for readability.
- Easy to be off by 1 iteration. Don't forget – there are $n+1$ ints between 0 and n inclusive. Check yourself with good borderline test cases.
- Make sure each iteration makes progress towards loop condition (or you might not terminate!).
- Don't forget the other caveats from earlier!

Exercises

1. Input a sequence of non-negative numbers using cin (terminated with a -1). Output the average of the even-indexed and odd-indexed ones.
   Ex: if the input is 3,7,2,18,7,8,-1, output 4 and 11.
2. With above sequence, output the average of the even and odd elements.
   Ex: with above data, output 5.67 and 9.33
3. You used one iteration construct above. Rewrite it (twice) to use the 2 other types of iteration constructs.
4. Input a positive integer k. Repeat the step: If k is even, divide it by 2; otherwise replace it with 3k+1. Print a success message iff the number eventually reaches 1 (infinite loop otherwise). The case of reaching 1 for all k is also called the Collatz conjecture (google it).
5. Input a positive integer n. Print a success message iff the above works for all integers k $\in$ [1,n].
   Hint: write a loop in which the previous code is nested.
6. Input a string s. Repeat the step: If s has even length, remove the right half of s; otherwise replace s with 3 copies of s followed by s's first character. Print a success message iff the string eventually has length 1. You may use the `substr` library function.
   Hint: how does the previous code change?