**5b. Loop statements**

**Topic: loop statements**
- The mathematics of loops
- Kinds of loops
  - sentinel controlled
  - counted
  - general exit tested
- C/C++ statements
  - while (top tested)
  - do...while (bottom tested)
  - for (counted, top tested)
- Unstructured statements
- File I/O loops
- Loop syntax

**The math of loops: Peano’s axioms**

1. 0 is a natural number
2. Every natural number $n$ has a unique successor, $n'$, also a natural number
3. All natural numbers follow (1) or (2)

- **Significance:** These axioms, or assumptions, provide a formal logical basis to work with counting numbers.
- Computation is a formal way to manipulate numbers and objects representable by them.

---

**Predecessors and addition**
- 1 is shorthand for $0'$ (successor of 0), 2 for $0''$, etc.
- Every natural number but 0 has a unique predecessor, $\text{pred}(n)$. Where $m' = n$, $m$ is the predecessor of $n$.
- $(a + b)$ is shorthand for $
 \begin{cases}
     a & \text{if } b = 0 \\
     a' + \text{pred}(b) & \text{otherwise}
 \end{cases}$
- Addition is an example of a function that is computable using a loop
- **Significance:** Recursively (inductively) definable sets and operations such as $\mathbb{N}$ and $+$, are precisely those that are computable. Any repetitive process can be specified by inductive methods.

---

**Example: $\Sigma$ (Sigma, summation)**
- The summation operator $\Sigma$ lets us add a series of numbers
- **Case:**
  $\sum_{k=1}^{n} k = \begin{cases}
    1 & \text{if } n = 1 \\
    n + \sum_{k=1}^{n-1} k & \text{otherwise}
  \end{cases}$
- E.g.: $\sum_{k=1}^{2} k = 1 + 2 + 3 = 6$
- **Generalizing:**
  $\sum_{k=1}^{n} f(k) = \begin{cases}
    f(1) & \text{if } n = 1 \\
    f(n) + \sum_{k=1}^{n-1} f(k) & \text{otherwise}
  \end{cases}$
- **Examples:** if $f(x) = 2$; if $f(x) = x$, etc.

---

**What triggers loop exit?**
- A loop may terminate (a) after a count, or (b) after a certain sentinel value is input
- Other kinds of exit condition are possible
- Counted loop:
  - Do 50 times:
    - draw a hyphen
  - Sentinel controlled loop:
    - Repeat
    - input a number until number is 0

---

**Top-tested loops: while**

A loop to count from 1 to 10:

```cpp
int i = 1;
while (i <= 10)
{
    cout << i << " ";
    i += 1;
}
```

**Output:**

1 2 3 4 5 6 7 8 9 10
5b. Loop statements

A sentinel-controlled loop

```cpp
const int sentinel = 0;
int input = -1, total = 0;
while (input != sentinel)
{
    cout << "Enter int \( \) \( \)\to exit\( \) \( \): \";
    cin >> input;
    if (input != sentinel)
        total += input;
    cout << \"Total: \" \( \) \( \)\total;\n}
```

- Danger with sentinels: users can confuse sentinel value with non-sentinel
- What if `year = 99` were sentinel in a file of student records?

Bottom-tested (do…while) loops

**Input validation:**

```cpp
int age;
do {
    cout << \"Age\? \";
    cin >> age;
} while (age < 0);
```

- The body of a bottom-tested loop will always execute at least once
- The loop above has a general exit condition
- Solve an “until” problem with a bottom-tested loop

The `for` loop statement

**Initialization exit test update**

```cpp
for (int i = 1; i <= 10; ++i)
{
    cout << i << \" \";
} Output: 1 2 3 4 5 6 7 8 9 10
```

- The `for` statement is a convenience; it is equivalent to a `while` loop, but exit test and update may be built into the header
- Syntax is flexible; e.g. `for (;;) is valid`
- `for` is top tested

Loop statements are interchangeable

```cpp
int counter = 1;
do {
    cout << counter \" \";
    ++counter;
} while (counter <= 10);
```

**Boolean variables for loop control**

```cpp
const int sentinel = 0;
bool done = false;
int total = 0;
while (done = = false)
{
    // <total> stores sum of input so far
    int input;
    cout << \"Enter a number, 0 to exit: \";
    cin >> input;
    if (input != sentinel)
        total += input;
    else
        done = true;
    cout << total;
```

- Boolean loop-control variable (flag)

An uncontrolled infinite loop hangs a program

- The hardest-to-find infinite loop is one that may exit sometimes:
  ```cpp
  int power;
  while (power < 1000)
  {
      cout << power \" \";
      power *= 2;
  }
  ```

- To exit, a loop must change a value that is tested in the exit condition:
  ```cpp
  int count = 0;
  while (count < 100)
  {
      cout << count \" \";
      ++count;
  }
  ```
Interactive computing uses controlled infinite loops

- Applications and operating environments today normally execute event-driven infinite loops:
  ```java
  do {
    event = get_user_input();
    respond_to(event);
  } while (event != Quit);
  ```

- Non-algorithmic interactive computation occurs when input and output are interleaved with termination determined by input

- Objects (topic 9) may respond infinitely to messages if never sent a “destroy” message

- Reference: P. Wegner, CACM, 5/97

Which statement to use?

- When the loop body should always execute, consider using `do...while` Example: User input and response to it
- When the loop body should sometimes not run, use `while` Example:
  ```java
  while (! infile.eof())...
  ```

Unstructured statements

- `break`: terminates branch or loop
- `return`: terminates a function
- `goto`: jumps to a named label
- `continue`: jumps to next iteration of a `for` loop
- `exit()`, `abort()`: terminate program

These statements support control structures other than the recommended sequence, branch, and loop

Middle-tested loops

(not recommended)

Example:
```java
while(true)
{
  int input;
  cin >> input;
  if (input == 0)
    break;
  cout << input;
}
```

Invariants help verify correctness

- Invariant: `sum` stores \( \sum_{i=1}^{f} \text{term}_i \)
- Postcondition of this pseudocode: `sum` stores \( \sum_{k=1}^{3} \text{term}_k \)

Convergence of exit-test value assures termination

- The value \( n \) converges on 0
- This guarantees that the loop will exit

- Example:
  ```java
  Input n
  count ← 0
  while n > 0
  n ← \lfloor n / 2 \rfloor
  count ← count + 1
  ```
5b. Loop statements

Kinds of errors

- Syntax: violation of grammar rule; caught by compiler
- Specification: Program solves the wrong problem
- Logic: Programmer’s chief concern; program produces incorrect or unpredictable output
- Runtime: Cause abnormal termination due to invalid operations, illegal memory access, etc. Preventable.

Common pitfalls with loops

- Declaring inside a loop a value updated by the loop
- Iterating one too few times
- Iterating one too many times
- Impossible exit conditions
  - value tested not changed in loop body
  - value changed may fail to move toward exit value
- Exit condition that never fails

Loop syntax

loop-statement:
  while ( expression ) statement
  do statement while ( expression ) ;
  for ( expr ; expr ; expr ) statement

Loop semantics

- The expression in the while and do...while statements is the exit test.
- The expressions in the for statement are for initialization, exit test, and updating.

What are the invariants?

```cpp
int input, total = 0;
icin >> input;
while (input > 0)
{
icin >> input;
total += input;
}
```

```cpp
int i = 0, count = 0;
char input[80];
cin.getline(s,80);
while (i < strlen(s))
{
  if (input[i] == ' ')
  count++;
  +i;
}
```

Tracing a loop

- When a loop produces bad results, tracing hidden values helps in debugging
- Trace statement below shows garbage values

```cpp
int count, input, total;
icin >> input;
while (input > 0)
{
cout << "input= " << input
<< " total= " << total << endl;
icin >> input;
total += input;
}
```