The goto statement allows transfer of control to any labelled point with a function. For example, this code:

```c
for (int i = 1; i <= 10; i++) {
    printf("%d\n", i);
}
```

can be written as:

```c
int i = 1;
loop:
    if (i > 10) goto end;
    i++;
    printf("%d", i);
    printf("\n");
    goto loop;
end:
```
• goto statements can result in very difficult to read programs.
• goto statements can also result in slower programs.
• In general, use of goto is considered poor programming style.
• Do not use goto without very good reason.
• kernel & embedded programmers sometimes use goto.
Writing correct assembler directly is hard.

Recommended strategy:

• develop the solution in C
• map to “simplified” C
• translate each simplified C statement to MIPS instructions

Simplified C

• does *not* have while, compound if, complex expressions
• *does* have simple if, goto, one-operator expressions
Simplified C makes extensive use of

- *labels* ... symbolic name for C statement
- *goto* ... transfer control to labelled statement

Example:

```
Standard C                                Simplified C
------------------  ------------------
i = 0; n = 0;
while (i < 5) {
    n = n + i;
    i++;  
}
```

```
i = 0; n = 0;
loop:
    if (i >= 5) goto end;
    n = n + i;
i++;  
goto loop;
end:
```
Mapping C into MIPS

Things to do:

- allocate variables to registers/memory
- place literals in data segment
- transform C program to:
  - break expression evaluation into steps
  - replace control structures by goto
- push function args explicitly
- implement function stack management
Example Printing First 10 Integers

```c
int main(void) {
    for (int i = 0; i <= 10; i++) {
        printf("%d\n", i);
    }
}
```
Example Printing First 10 Integers

Convert to goto and simple C statements and decide where variables will be stored.

```c
int main(void) {
    int i; /* in register \$t0, */
    i = 0;
    loop:
        if (i >= 10)
            goto end;
        i++;
        printf("%d", i);
        printf("\n");
        goto loop;
    end:
        return 0;
}
```
int main(void) {
    int i; // in register $t0

    li $t0, 0  // i = 0;

    loop:
        bge $t0, 10 end  // if (i >= 10) goto end;
        add $t0, $t0 1  // i++;
        move $a0, $t0  // printf("%d" i);
        li $v0, 1
        syscall
        la $a0, newline  // printf("\n");
        li $v0, 4
        syscall
        b loop  // goto loop;

    end:
        jr $ra  // return

.data

newline:
    .ascii "\n"