goto in C

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

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

can be written as:

```
    int i = 1;
loop:
    if (i > 10) goto end;
        i++;
        printf("%d", i);
        printf("\n");
    goto loop;
end:
```

goto in C

- **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 **bad** programming style.
- Do not use **goto** without very good reason.
- kernel & embedded programmers sometimes use goto.

MIPS Programming

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:

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

adding 2 numbers: C to simplified C

C

```c
int main(void) {
    int x = 17;
    int y = 25;
    printf("%d\n", x + y);
    return 0;
}
```

source code for add.c

Simplified C

```c
int main(void) {
    int x, y, z;
    x = 17;
    y = 25;
    z = x + y;
    printf("%d", z);
    printf("\n");
    return 0;
}
```

source code for add.simple.c

adding 2 numbers: simplified C to MIPS

Simplified C

```
int x, y, z;
x = 17;
y = 25;
z = x + y;
printf("%d", z);
printf("\n");
```

MIPS

```
# add 17 and 25  and print result
main:                       # x,y,z in $t0,$t1,
    li   $t0, 17            # x = 17;
    li   $t1, 25            # y = 25;
    add  $t2, $t1, $t0 # z = x + y
    move $a0, $t2           # printf("%d", z);
    li   $v0, 1
    syscall
    li   $a0, '\n'          # printf("%c", '\n
    li   $v0, 11
    syscall
    li   $v0, 0             # return 0
    jr   $ra
```

source code for add.s

while loop - converting C to simplified C

Standard C

```
i = 0;
n = 0;
while (i < 5) {
    n = n + i;
    i++;
}
```

Simplified C

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

while loop - converting simplified C to MIPS

Simplified C

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

MIPS

```
li $t0, 0   # i in $t0
li $t1, 0   # n in $t1
loop:
  bge $t0, 5, end
  add $t1, $t1, $t0
  add $t0, $t0, 1
  goto loop
end:
```

if - converting C to simplified C

Standard C

```
if (i < 0) {
    n = n - i;
} else {
    n = n + i;
}
```

Simplified C

```
if (i >= 0) goto else1;
    n = n - i;
    goto end1;
else1:
    n = n + i;
end1:
```

- note `else` can't be used a a label in C

if - converting simplified C to MIPS

Simplified C

```
if (i >= 0) goto else1;
    n = n - i;
    goto end1;
else1:
    n = n + i;
end1:
```

MIPS

```
# assume i in $t0
# assume n in $t1
  bge $t0, 0, else1
  sub $t1, $t1, $t0
  goto end1
else1:
  add $t1, $t1, $t0
end1:
```

if/and: C to simplified C

Standard C

```
if (i < 0 && n >= 42) {
    n = n - i;
} else {
    n = n + i;
}
```

Simplified C

```
if (i >= 0) goto else1;
if (n < 42) goto else1;
    n = n - i;
    goto end1;
else1:
    n = n + i;
end1:
```

if/and: simplified C to MIPS

Simplified C

```
if (i >= 0) goto else1;
if (n < 42) goto else1;
    n = n - i;
    goto end1;
else1:
    n = n + i;
end1:
```

MIPS

```
# assume i in $t0
# assume n in $t1
  bge $t0, 0, else1
  blt $t1, 42, else1
  sub $t1, $t1, $t0
  goto end1
else1:
  add $t1, $t1, $t0
end1:
```

odd-even: C to simplified C

Standard C

```
if (i < 0 || n >= 42) {
    n = n - i;
} else {
    n = n + i;
}
```

Simplified C

```
if (i < 0) goto then1;
if (n >= 42) goto then1;
goto else1;
then1:
    n = n - i;
    goto end1;
else1:
    n = n + i;
end1:
```

Printing First 10 Integers: C to simplified C

C

```c
int main(void) {
    for (int i = 1; i <= 10; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

source code for print10.c

Simplified C

```c
int main(void) {
    int i;
    i = 1;
loop:
    if (i > 10) goto end;
        i++;
        printf("%d", i);
        printf("\n");
    goto loop;
end:
    return 0;
}
```

source code for print10.simple.c

Printing First 10 Integers: MIPS

```
# print integers 1..10 one per line
main:                   # int main(void) {
                        # int i;  // in register $t0
    li    $t0, 1        # i = 1;
loop:                   # loop:
    bgt   $t0, 10 end   # if (i > 10) goto end;
    move $a0, $t0       #   printf("%d" i);
    li    $v0, 1
    syscall
    li    $a0, '\n'     # printf("%c", '\n');
    li    $v0, 11
    syscall
    add   $t0, $t0, 1   #   i++;
    b     loop          # goto loop;
end:
    li    $v0, 0        # return 0
    jr    $ra
```

Odd or Even: C to simplified C

C

```c
int main(void) {
    int x;
    printf("Enter a number: ");
    scanf("%d", &x);
    if ((x & 1) == 0) {
        printf("Even\n");
    } else {
        printf("Odd\n");
    }
    return 0;
}
```

source code for odd_even.c

Simplified C

```c
int main(void) {
    int x, v0;
    printf("Enter a number: ");
    scanf("%d", &x);
    v0 = x & 1;
    if (v0 == 1) goto odd;
        printf("Even\n");
    goto end;
odd:
        printf("Odd\n");
end:
    return 0;
}
```

source code for odd_even.simple.c

Odd or Even: MIPS

```
# read a number and print whether its odd or even
main:
    la   $a0, string0     # printf("Enter a number: ");
    li   $v0, 4
    syscall
    li   $v0, 5           # scanf("%d", x);
    syscall
    and  $t0, $v0, 1      # if (x & 1 == 0) {
    beq  $t0, 1, odd
    la   $a0, string1     # printf("Even\n");
    li   $v0, 4
    syscall
    b end
```

source code for odd_even.s

Odd or Even: MIPS

```
odd:                        # else
    la   $a0, string2       # printf("Odd\n");
    li   $v0, 4
    syscall
end:
    li   $v0, 0             # return 0
    jr   $ra
    .data
string0:
    .asciiz "Enter a number: "
string1:
    .asciiz "Even\n"
string2:
    .asciiz "Odd\n"
```

source code for odd_even.s

Sum 100 Squares: C to simplified C

C

```c
int main(void) {
    int sum = 0;
    for (int i = 0; i <= 100; i++) {
        sum += i * i;
    }
    printf("%d\n", sum);
    return 0;
}
```

source code for sum_100_squares.c

Simplified C

```c
int main(void) {
    int i, sum, t3;
    sum = 0;
    i = 0;
    loop:
        if (i > 100) goto end;
        t3 = i * i;
        sum = sum + t3;
        i = i + 1;
    goto loop;
end:
    printf("%d", sum);
    printf("\n");
    return 0;
}
```

source code for sum_100_squares.simple.c

Sum 100 Squares: MIPS

```
# calculate 1*1 + 2*2 + ... + 99 * 99 + 100 * 100
# sum in $t0, i in $t1
main:
    li  $t0, 0          # sum = 0;
    li  $t1, 0          # i = 0
loop:
    bgt $t1, 100 end    # if (i > 100) goto end;
    mul $t3, $t1, $t1   # t3 = i * i;
    add $t0, $t0, $t3   # sum = sum + t3;
    add $t1, $t1, 1     # i = i + 1;
    b   loop
end:
```

source code for sum_100_squares.s

Sum 100 Squares: MIPS

```
end:
    move $a0, $t0       # printf("%d", sum);
    li   $v0, 1
    syscall
    li   $a0, '\n'       # printf("%c", '\n');
    li   $v0, 11
    syscall
    li   $v0, 0          # return 0
    jr   $ra
```

source code for sum_100_squares.s