CPNM Lecture 4 - C Control Statements

Mridul Sankar Barik

Jadavpur University

2022

C Control Statements

1. Three categories
   - **Selection statement** - allow a program to select a particular execution path, ex: `if`, `switch`
   - **Iteration statements** - support iterations, ex: `while`, `do`, `for`
   - **Jump statement** - cause an unconditional jump to some other place in the program, ex: `break`, `continue`, `goto`

if Statement I

- Allows a program to choose between two alternatives by testing the value of an expression
  `if ( expression ) statement`
- A nonzero value of the expression is treated as `true`
- Compound Statements: a group of statements within braces
  `if ( expression ) { statements }`
- The `else` clause
  `if ( expression ) { statements } else { statements }`
- Example

```
if (i > j )
    max = i;
else
    max = j;
```

if Statement II

- Nested `if`

```
if (i > j) {
    if (i > k)
        max = i;
    else
        max = k;
}
else{
    if(j > k)
        max = j;
    else
        max = k;
}
```

- Braces can be added even when they're not necessary

if Statement III

- Cascaded if statement: used to test a series of conditions, stopping as soon as one of them is true.

```
if (n < 0)
    printf("n is less than 0\n");
else
    if (n == 0)
        printf("n is equal to 0\n") ;
    else
        printf("n is greater than 0\n");
```

Iteration Statements I

- C's iteration statements allows us to set up loops
- A loop is a statement whose job is to repeatedly execute some other statement (the loop body)
- Every loop has a controlling expression. Each time the loop body is executed (an iteration of the loop), the controlling expression is evaluated; if the expression is true-has a value that's not zero - the loop continues to execute
- C provides three iteration statements: `while, do, for`
- The `while` statement is used for loops whose controlling expression is tested before the loop body is executed
- The `do` statement is used if the expression is tested after the loop body is executed
- The `for` statement is convenient for loops that increment or decrement a counting variable

The while Statement I

- `while (expression ) statement`
- Controlling expression is evaluated first. If its value is non zero (true), the loop body is executed and the expression is tested again
- Example: Print "Hello" 5 times

```
i = 1;
while ( i <= 5 ){ /* Controlling expression */
    printf("Hello\n"); /* Loop body */
    i++;
}
```

- The controlling expression is false when a while loop terminates
- Infinite loops: A while statement won't terminate if the controlling expression always has a nonzero value

The while Statement II

```
while (1) {
    printf("Hello\n");
    ...
}
```

- Example: Printing a table of squares

```
#include<stdio.h>
int main (void){
    int i, n;
    printf ("This program prints a table of squares: \n");
    printf ("Enter number of entries in table: ");
    scanf ("%d", &n);
    i = 1;
    while (i <= n){
        printf("%10d%10d\n", i , i*i);
        i++;
    }
    return 0;
```

The while Statement III

```
}
```

- Example: Summing a series of numbers

```
#include <stdio .h>

int main (void){
    int n, sum = 0;
    printf("This program sums a series of integers. \n");
    printf("Enter integers (0 to terminate): ");
    scanf("%d", &n);
    while (n != 0) {
        sum = sum + n;
        scanf("%d", &n);
    }
    printf("The sum is: %d\n", sum);
    return 0;
}
```

The do Statement I

- Essentially it is just a `while` statement whose controlling expression is tested after each execution of the loop body
- `do { statements } while(expression);`
- Example: Calculating the number of digits in an integer

```
#include <stdio.h>

int main(void){
    int digits=0, n;
    printf("Enter a nonnegative integer: ");
    scanf("%d", &n);
    do {
        n = n / 10;
        digits++;
    } while (n > 0);
    printf("The number has %d digit(s) \n", digits);
    return 0;
}
```

The do Statement II

- If we replace the do loop by a similar while loop:

```
while (n > 0){
    n = n / 10;
    digits++;
}
```

If n is 0 initially, this loop won't execute at all, and the program would print `The number has 0 digit(s)`

The for Statement I

- Format: `for (expr1 ; expr2 ; expr3) statement`
- Closely related to `while` statement

```
expr1;
while ( expr2 ) {
    statement;
    expr3;
}
```

- Best choice for loops that "count up" (increment a variable) or "count down" (decrement a variable)
- Omitting Expressions in a for Statement

```
i = 10;
for (;i > 0; i--){
    printf("Hello\n");
}
```

The for Statement II

```
for (i = 10; ; i--){
    printf("Hello\n");
}
```

```
for (i = 10; i > 0; ){
    printf("Hello\n");
}
```

- We can use a comma expression as the first or third expression in the for statement - `expr1, expr2`

```
for ( i = 0, j = 10; i != j ; i++, j--){
    printf("Hello\n");
}
```

Declarations within for Loop

```
int i=0, j=10;
for(int i=0, j=10; i!=j; i++,j--){
    int i=0, j=10;
    printf("Hello ");
    printf("i=%d, j=%d\n", i, j);
}
printf("i=%d, j=%d\n", i, j);
```

Output:

```
Hello i=0 j=10
Hello i=0 j=10
Hello i=0 j=10
Hello i=0 j=10
Hello i=0 j=10
i=0 j=10
```

When we unfold the loop it becomes:

```
int i=0, j=10;
{
    int i=0, j=10;
    label1: if(i!=j)
    {
        int i=0, j=10;
        printf("Hello ");
        printf("i=%d, j=%d\n", i, j);
    }
    else goto label2;
    i++; j--;
    goto label1;
}
label2: printf("i=%d, j=%d\n", i, j);
```

Exiting from a Loop I

- The `break` statement: it can be used to jump out of a `while`, `do`, or `for` loop.
- Example: To check whether a number `n` is prime

```
for (d = 2; d < n; d++){
    if (n % d == 0)
        break;
}
if (d < n)
    printf("%d is divisible by %d\n", n, d);
else
    printf("%d is prime\n", n);
```

- A `break` statement transfers control out of the innermost enclosing `while`, `do`, `for`, or `switch` statement
- The `continue` statement: `break` transfers control just past the end of a loop, while `continue` transfers control to a point just before the end of the loop body

Exiting from a Loop II

- Use of `continue` statement is limited to loops
- Example: Find sum of 10 non zero numbers to be read from the user

```
n = 0;
sum = 0:
while (n < 10) {
    scanf("%d", &i);
    if (i == 0 )
        continue;
    sum += i:
    n++;
    /* continue jumps here */
}
```

The goto Statement

- goto statement allows jumping to any statement in a function, provided that the statement has a label
- A label is just an identifier placed at the beginning of a statement:

```
identifier:  statement
goto identifier;
```

- Example:

```
for (d = 2; d < n; d++){
    if (n % d == 0)
        goto done:
}
done: if (d < n)
        printf("%d is divisible by %d\n", n, d);
      else
        printf("%d is prime\n", n);
```

The null Statement

- A statement can be null - devoid of symbols except for the semicolon at the end
- Suppose we need to put a label at the end of a compound statement. A label can't stand alone: it must always be followed by a statement. Putting a null statement after the label solves the problem.

```
{
    ...
    goto end_of_stmt;
    ...
    end_of_stmt: ;
}
```

The switch Statement I

- Used when we need to compare an expression against a series of values to see which one it currently matches
- Format

```
switch ( expression ) {
    case constant-expression : statements
    ...
    case constant-expression : statements
    default: statements
}
```

- A constant expression is much like an ordinary expression except that it can't contain variables or function calls
- The constant expression in a case label must evaluate to an integer (characters are also acceptable).
- No braces are required around the statements
- The last statement in each group is normally `break`
- Duplicate case labels aren't allowed. The order of the cases doesn't matter.

The switch Statement II

- A float expression cannot be tested using a `switch`
- Example

```
switch(grade){
    case 4: printf("Excellent"); break;
    case 3: printf("Good"); break;
    case 2: printf("Average"); break;
    case 1: printf("poor"); break;
    case 0: printf("Failing"); break;
    default: printf("Illegal grade"); break;
}
```

- There's no way to write a case label that specifies a range of values.
- A `switch` statement isn't required to have a `default` case
- A `switch` statement is often easier to read than a cascaded if statement
- `switch` statements are often faster than if statements
  - `switch-case` is often implemented using a jump table with the case values as index into the table

The Switch Statement - Example I

```
/* Prints a date in legal form */

#include <stdio.h>
int main (void){
    int month, day, year;

    printf("Enter date (mm/dd/yy): ");
    scanf("%d / %d / %d", &month, &day, &year);
    printf ("Dated this %d", day);

    switch(day){
        case 1: case 21: case 31:
        printf ("st"); break;
        case 2: case 22:
        printf("nd"); break;
        case 3: case 23:
        printf("rd"); break;
        default: printf("th"); break;
    }

    printf(" day of ");

    switch(month){
```

The Switch Statement - Example II

```
        case 1: printf ("January"); break;
        case 2: printf("February"); break;
        case 3: printf("March"); break;
        case 4: printf("April"); break;
        case 5: printf("May"); break;
        case 6: printf("June"); break;
        case 7: printf("July"); break;
        case 8: printf("August"); break;
        case 9: printf("September"); break;
        case 10: printf("October"); break;
        case 11: printf("November"); break;
        case 12: printf("December"); break;
    }
    printf(", 20%.2d.\n", year);
    return 0;
}
```

Case Ranges - C Language Extension

- GNU C provides several language features not found in ISO standard C
  - It supports, as a language extension, case ranges
- The `-pedantic` option directs GCC to print a warning message if any of these features is used
- Be careful: Write spaces around the ...

```
#include <stdio.h>
main() {
   int data[10] = { 5, 4, 10, 25, 60, 47, 23, 80, 14, 11};
   int i;
   for(i = 0; i < 10; i++) {
      switch (data[i]) {
         case 1 ... 10:
            printf("%d in range 1 to 10\n", data[i]);
         break;
         case 11 ... 20:
            printf("%d in range 11 to 20\n", data[i]);
         break;
         case 21 ... 30:
            printf("%d in range 21 to 30\n", data[i]);
         break;
         case 31 ... 40:
            printf("%d in range 31 to 40\n", data[i]);
         break;
         default:
            printf("%d Exceeds the range\n", data[i]);
         break;
      }
   }
}
```