Flow Control

Outline

❖ Blocks and compound statements
❖ Conditional statements
   ○ if - statement
   ○ if-else - statement
   ○ switch - statement
   ○ ? : opertator
   ○ Nested conditional statements
❖ Repetitive statements
   ○ for - statement
   ○ while - statement
   ○ do-while - statement
   ○ Nested repetitive statements
   ○ Break and continue statements
❖ Unconditional jump: `goto`

Blocks and Compound Statements

❏ A simple statement ends in a semicolon: `z = foo(x+y);`

❏ Consider the multiple statements:

```
temp = x+y ;
z = foo (temp) ;
```

- Curly braces – combine into compound statement/block
- Block can substitute for simple statement
- Compiled as a single unit
- Variables can be declared inside
- No semicolon at end

```
{
   int temp = x+y;
   z = foo(temp);
}
```

❏ Block can be empty {}

Blocks and Compound Statements

❏ Blocks nested inside each other

```
{
  int temp = x+y ;
  z = foo ( temp ) ;
  {
    float temp2 = x*y ;
    z += bar ( temp2 ) ;
  }
}
```

❏ Variables declared inside a block are only visibly within this block and its internatl blocks

Conditional Statements

- ❏ **if** - Statement

- ❏ **if-else** - Statement

- ❏ **switch** - Statement

- ❏ **? :** Ternary operator

- ❏ No boolean type in ANSI C
  - ○ introduced in C99

- ❏ Relational and logical expressions are evaluated to:
  - ○ `1` if they are logically true
  - ○ `0` if they are logically false

- ❏ Numeric expressions are considered false if they are evaluated to integer 0

- ❏ Pointer expressions are considered false if they are evaluated to null

if- Statement

❏ Syntax:

```
if (<condition>)
    <statement>;
```

❏ Example:

```
if ( x % 2 == 0)
   y += x / 2 ;
```

- Evaluate condition: `(x % 2 == 0)`
  - If true, execute inner statement: `y += x/2;`
  - Otherwise, do nothing
- Inner statements may be block

if-else - Statement

❏ Syntax:
```
if (<condition>)
  <statement1>;
else
  <statement2>;
```

❏ Example:
```
if ( x % 2 == 0)
  y += x / 2 ;
else
  y += ( x + 1 ) / 2;
```

○ Evaluate condition: `(x % 2 == 0)`
■ If true, execute first statement: `y += x/2;`
■ Otherwise, execute second statement: `y += ( x + 1 ) / 2;`

○ Either inner statements may be block

Nesting if/if-else Statements

- ❏ Can have additional alternative control paths by nesting `if` statements:
  ```
  if (<condition>)
    <statement1>; /* can be an if or if-else statement*/
  else
    <statement2>; /* can be an if or if-else statement*/
  ```
- ❏ Conditions are evaluated in order until one is met; inner statement then executed
  - ○ if multiple conditions true, only first executed
- ❏ Example:
  ```
  if ( x % 2 == 0)

    y += x / 2 ;

  else if ( x % 4 == 1)

    y += 2 * (( x + 3 )/ 4 );

  else

    y += ( x +1 )/ 2 ;
  ```

Nesting if/if-else Statements

❏ **Dangling `else`** , example:

```
if ( x % 4 == 0)
if ( x % 2 == 0)
y = 2;
else
y = 1;
```

```
if ( x % 4 == 0)
  if ( x % 2 == 0)
    y = 2;
  else
    y = 1;
```

```
if ( x % 4 == 0)
  if ( x % 2 == 0)
    y = 2;
else
  y = 1;
```

- ○ To which if statement does the else keyword belong?
  Belongs to the nearest if in the same block
- ○ To associate else with outer if statement: use braces

```
if ( x % 4 == 0) {
  if ( x % 2 == 0)

    y = 2;
} else
    y = 1;
```

switch - Statement

❏ Syntax:
```
switch (<int or char expression>) {
  case <literal1>:    <statements>
                      [break;]

  [more cases]
  [default:  <statements>]
}
```
❏ Provides multiple paths
❏ Case labels: different entry points into block
❏ Compares evaluated expression to each case in order:
  ○ When match found, starts executing inner code until `break;` reached
  ○ Execution "falls through" if `break;` is not included

switch - Statement

❏ Example:

```
switch ( ch ) {
  case 'Y' : /* ch == 'Y ' */
             /* do something */
             break ;
  case 'N' : /* ch == 'N ' */
             /* do something else */
             break ;
  default :  /* otherwise */
             /* do a third thing */
}
```

Loops (Iterative Statements)

- ❏ **while** - loop
- ❏ **for** - loop
- ❏ **do-while** - loop
- ❏ **break** and **continue** keywords

Loops: while - Statement

- ❏ Syntax:
  ```
  while ( <condition> )
   <loop body>
  ```

- ❏ Simplest loop structure – evaluate body as long as condition is true
- ❏ Condition evaluated first, so body may never be executed
- ❏ Example:

Loops: for - Statement

❏ Syntax:
```
for ([<initialization>];[<condition>];[<modification>])
   <loop body>
```

❏ Example:
```
int i , j = 1;
for ( i = 1; i <= n ; i ++)
  j *= i ;
printf("%d\n", j);
```

- ○ A "counting" loop
- ○ Inside parentheses, three expressions, separated by semicolons:
    - ■ Initialization: i = 1
    - ■ Condition: i <= n
    - ■ Modification: i++

Loops: for - Statement

❏ Any expression can be empty (condition assumed to be "true"):
```
for (;;) /* infinite loop */
  <loop body>
```
❏ Compound expressions separated by commas
- Comma: operator with lowest precedence, evaluated left-to-right
```
for ( i = 1 , j = 1; i <= n ; j  *= i , i ++)
  <loop body>
```
❏ Equivalent to while loop:
```
<initialization>
while (<condition>) {
  <loop body>
  <modification>
}
```

Loops: do-while - Statement

- ❏ Syntax:
```
do {
    <loop body>
} while(<condition>);
```
- ❏ Differs from while loop – condition evaluated after each iteration
  - ○ Body executed at least once
  - ○ Note semicolon at end
- ❏ Example:
```
char c ;
do {
/ * loop body * /
puts( "Keep going? (y/n) " ) ;
c = getchar();
/ * other processing * /
} while ( c == 'y' && /* other conditions */ );
```

Loops: Nested Loops

- ❏ A nested loop is a loop within a loop
    - ○ an inner loop within the body of an outer one.
      ```
      for ([<initialization>];[<condition>];[<modification>])
        <loop body> /* another loop here */
      ```

- ❏ Can nest any loop statement within the body of any loop statement

- ❏ Can have more than two levels of nested loops

Loops: break - Statement

- ❏ Sometimes want to terminate a loop early
  - ○ break; exits innermost loop or switch statement to exit early
  - ○ Consider the modification of the do-while example:

```
char c ;
do {
  /* loop body */
  puts ( "Keep going? (y/n) " ) ;
  c = getchar() ;
  if ( c != 'y')
    break ;
  /* other processing */
} while ( /* other conditions */ ) ;
```

Loops: continue - Statement

❏ Use to skip an iteration
  ○ continue; skips rest of innermost loop body, jumping to loop condition

❏ Example:

```
int i , ret = 1 , minval;
for ( i = 2; i <= (a > b? a:b); i++) {
  if ( a % i ) /* a not divisible by b */
    continue;
  if ( b % i == 0) /* b and a are multiple of i */
    ret = i;
}
printf("%d\n", ret);
```

Unconditional Jump

- ❏ `goto`: transfers program execution to a labeled statement in the current function
  - ○ DISCOURAGED
  - ○ easily avoidable
  - ○ requires a label

- ❏ Label: a plain text, except C keywords, followed by a colon, prefixing a code line
  - ○ may occur before or after the `goto` statement

- ❏ Example:
```c
int main () {
    int a = 10;
    LOOP:do {
         if ( a == 15) {
           a = a + 1;
           goto LOOP;
         }
         printf("value of a: %d\n", a++);
    } while( a < 20 );
   return 0;
```