Lecture 3: Statements and Expressions
Statements
Overview

- Statements are used to realize certain tasks in a program: assignments, calculations, data input/output, function calls, …

- Types:
  - Simple statements
    - `printf("Hello!");`
    - `x = PI*R*R;`
  - Compound statement `{ … }`
  - Control structures (loops, conditional branches): for, if, while, switch, break,…
  - Special statements: null, expression, label, return,…
if statements

- **Conditional branches**

- **Syntax:**
  - \( \text{if } (<\text{condition}>) \ <\text{statement}> \)
    - [else <statement>]

- **Ex:**
  - if (x != 0.)
    - printf("Inverse = %f",1/x);
    - else printf("Irreversible");

  - if (score > current_record)
    - NewRecord(score);
Nested if statements

- Nested if statements are usually used to check multiple conditions

Ex:

```c
if (mark >= 8.)
    printf("Good grade");
else if (mark >= 7.)
    printf("Fairly good");
else if (mark >= 5.)
    printf("Passed");
else
    printf("Not passed");
```
Logical expressions

- Are expressions that contain logical operators, variables or constants
- Evaluated to 1 (true) or 0 (false)
- In C, int can be implicitly interpreted as logical type with following conversion: 0 → false, other than 0 → true
  - Consequence: comparing an integer with 0 can be ignored in logical expressions:
    - if (x != 0) ...
    - → if (x) ...
- Ex:
  - 8*4 >= 10
  - x != y
  - b*b > 4*a*c
  - (a>2) && ((b<3) || (a>4)))
  - 2-3 /* implicitly interpreted as true */
switch statements

- Conditional branches

- Syntax:
  - `switch (<expression>) {`  
  - `case <value 1>: <statements>`
  - `case <value 2>: <statements>`
  - `...`
  - `[default: <statements>]`
  - `}

- Execute different statements depending on which case the expression value corresponds to

- default case is executed when the expression value corresponds to none of the above cases

- The execution falls to the next case after finishing one → use break to terminate if desired

- Can be only used with integer expressions (char, int, enum,…), and the listed values must be constant
switch statements (cont.)

- switch (x) {
  case 0:
    printf("x is 0");
    break;
  case 1:
    printf("x is 1");
    break;
  default:
    printf("x is something else");
}

- switch (day) {
  case Sat:
  case Sun:
    printf("Weekend");
    break;
  default:
    printf("Working");
}
do and while loops

- Used to repeatedly execute tasks

- Syntax:
  - while(<condition>) <statement>
    - Check the condition **before** execution of the statement in each iteration
  - do <statement> while(<condition>);
    - Check the condition **after** execution of the statement in each iteration

- Ex:
  - x = 0;
    do
      printf("%d ", x++);
    while (x < 10);
  - x = 0;
    while (x < 10)
      printf("%d ", x++);

Result:
0 1 2 3 4 5 6 7 8 9
for loops

- Very popular in C because of short syntax and flexibility

Syntax:

```c
for (<init-expr>; <cond-expr>; <increment-expr>)
    <statement>
```

- Each expression can be ignored when unused
- Condition is checked before each iteration

Ex:

```c
for (x = 0; x < 10; x++)
    printf("%d ", x);
```

Result:

```
0 1 2 3 4 5 6 7 8 9
```

```c
for (i = 0; i < N; i++)
    for (j = 0; j < M; j++)
        a[i][j] = i + j;
```
break and continue statements

- **break**: used to terminate a loop without checking for given condition
  
  ```c
  for (x = 1; x <= 10; x++) {
    if (x == 8) break;
    printf("%d ", x);
  }
  ```

- **continue**: used to terminate one iteration, and execute the next one
  
  ```c
  for (x = 1; x <= 10; x++) {
    if (x == 8) continue;
    printf("%d ", x);
  }
  ```
**block statements**

- Used to group multiple simple sub-statements into one, by using `{ ... }`
- Usually used in control structures
- Ex:
  ```c
  if (delta > 0) {
      d = sqrt(delta);
      x1 = (-b+d)/2.;
      x2 = (-b-d)/2.;
      printf("x1 = %.3f, x2 = %.3f", x1, x2);
  }
  ```
  
  ```c
  odd_sum = 0;
  sum = 0;
  for (i = 0; i < N; i++) {
      if (i%2 == 1) odd_sum += a[i];
      sum += a[i];
  }
  ```
Null statements

- Empty statement, just “;”
- Does not execute anything nor produce any effect, and does not take any execution time
- Used sometimes to finish control structures correctly in syntax: loops, conditional branches, labels,…

Ex:

```c
for (i=0; i<N; a[i]=b[i++]) ;
for (i=0; i<10; printf("%d ",i++)) ;
if (x==y) ;
else x=y;
```
goto statements and labels

- Label: used to mark a position in code
- `goto` statement: jump to a given certain label and continue execution
  ```
  printf("Day: ");
  scanf("%d", &day);
  if (day<1 || day>31) goto error;
  printf("Month: ");
  scanf("%d", &month);
  if (month<1 || month>12) goto error;
  /* ... */
  error:
  printf("Input error!");
  ```
- There must be at least one statement after a label (null statement can be used)
- `goto` statements destroy structurability → avoid to use
Expressions
Overview

- An expression contains at least one operands and 0, 1 or many operators
- Operand can be: value, constant, variable, result of a function or sub-expression
- Operators have precedence (order of evaluation), but can be modified by using ( … )
- Operators of same precedence are evaluated from left to right: sin(x) + cos(x) → sin(x) is evaluated before cos(x)

Ex:
- 47
- i
- x++
- sin(x+2)
- \( ( 2 \times \log( ( 3 + 10 ) - ( 2 \times 6 ) ) ) \)
Arithmetic operators

- **Basic:** 
  - \( a + b \)
  - \( a - b \)
  - \( a \times b \)
  - \( a / b \)

- Multiplication/division takes precedence over addition/subtraction: \( 2 + 3 \times 4 + 1 \)
- Operators on 2 integers \( \rightarrow \) integer results
- Operators on at least 1 floating-point \( \rightarrow \) floating-point

- **Modulo (integer remainder):** \( a \% b \)
- **Increment, decrement by 1:** \( a++ \quad a-- \quad ++a \quad --a \)
  - Operands must be integer variables
  - \( a++ \) and \( a--: \) expression is evaluated before changing the value of \( a \), while for \( ++a \) and \( --a \), the order is inversed
    - \( a++ \times 2 \)
    - \( ++a \times 2 \)
    - \( a++ + a++ \quad /* \text{avoid to use} */ \)
Logical and comparison operators

- **Logical:**
  - and: `A && B`
  - or: `A || B`
  - not: `!A`

- **Comparison:**
  - `x == y`
  - `x != y`
  - `x > y`
  - `x < y`
  - `x >= y`
  - `x <= y`
Bitwise operators

- **Including:**
  - **and:** \( a \& b \)
  - **or:** \( a \mid b \)
  - **xor:** \( a \^ b \)
  - **not:** \( \sim a \)
  - **Left/right shift:** \( a<<b \quad a>>b \)

- Applicable only to integer operands
- **Attn:** do not confuse with logical operators
Assignment operators

- **Basic:** \( a = b \)
- **Combined assignments:**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Equivalent</th>
<th>Operator</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a += b )</td>
<td>( a = a + b )</td>
<td>( a &amp;= b )</td>
<td>( a = a &amp; b )</td>
</tr>
<tr>
<td>( a -= b )</td>
<td>( a = a - b )</td>
<td>( a</td>
<td>= b )</td>
</tr>
<tr>
<td>( a *= b )</td>
<td>( a = a * b )</td>
<td>( a ^= b )</td>
<td>( a = a ^ b )</td>
</tr>
<tr>
<td>( a /= b )</td>
<td>( a = a / b )</td>
<td>( a &lt;&lt;= b )</td>
<td>( a = a &lt;&lt; b )</td>
</tr>
<tr>
<td>( a %= b )</td>
<td>( a = a % b )</td>
<td>( a &gt;&gt;= b )</td>
<td>( a = a &gt;&gt; b )</td>
</tr>
</tbody>
</table>

- Assignment returns a value is equal to the assigned one
  - \( x = y = 15; \)
  - \( \text{if } ((c = \text{getchar}()) == 'y' || c == 'n') \{ \ldots \} \)
  - \( \text{if } (a = b) \{ \ldots \} /* \text{compiled but incorrect */} \)
Other special operators

- Unary plus/minus: +a, -a
- sizeof: returns size of types/variables/constants
- Parentheses (): (a+b)*c
- Comma “,”: does not do anything, just discards the first (left) value
  - x++, y = x*2;
  - for (i=2, j=3, x=1; i<10; i+=2, j+=3, x++) {...}
- Indirection (value pointed by): *a
- Reference (address of): &a
- Structure reference/dereference: a.name, b->age
- Type cast: (int)a
- Function call (): sin(x), pow(x, 3)
- Ternary conditional: <condition> ? <expr 1> : <expr 2>
  - biggest = a > b ? a : b;
Problems

Write programs to:

1. Enter a, b, c and solve for 2\textsuperscript{nd} order equation

2. Calculate sum of inverses of even numbers from 2 to 100: \( \frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \cdots + \frac{1}{100} \)

3. Enter a chain of numbers and calculate the average value

4. Rewrite the following code using \textbf{while}

   \begin{verbatim}
   for (A; B; C) do_something();
   \end{verbatim}

5. Print a menu and read user’s choice by two ways: (1) using conditional branches, (2) using array

6. Calculate the traversed distance of a falling object at moments: \( t = 1, 2, 3, \ldots, 20 \)s

7. Same as Prob. 6, but enter the initial height and calculate only until the object reaches to ground