Tin học cơ sở 4

Control Flow
Outline

• Making choices/Branching:
  – The if construct
  – if-else
  – switch

• Repetition:
  – while
  – do-while
  – for
The *if* construct

The if statement allows choice between two execution paths. One form:

```
if (expression){
    statement
}
```

- Used to decide if `statement` should be executed.
- There is no explicit boolean type in C
  - In C: zero is regarded as “false”, non-zero is regarded as “true”
- `statement` is executed if the evaluation of `expression` is true.
- `statement` is NOT executed if the evaluation of `expression` is false.
- `statement` could be a single instruction, or a series of instructions enclosed in `{ }` – **always** use `{}`
The *if* construct (cont.)

Another form:

```plaintext
if (expression){
    statement1

}else {
    statement2
}
```

- Used to decide if *statement1 or statement2* should be executed.
- *statement1* is executed if the evaluation of *expression* is true.
- *statement2* is executed if the evaluation of *expression* is false.
The *if* construct example

Here is an example

```c
int x;
printf("x = ");
scanf("%i", &x);
if (x){
    printf(" x is non-zero");
} else {
    printf(" x is zero");
}
```
Style

- As you can see from the code examples, indentation is very important in promoting the readability of the code.
- Each logical block of code is indented.
- Each ‘{’ and ’}’ are indented to the appropriate logical block level.

<table>
<thead>
<tr>
<th>Style 1</th>
<th>Style 2 (preferred)</th>
</tr>
</thead>
<tbody>
<tr>
<td>if(x)</td>
<td>if (x){</td>
</tr>
<tr>
<td>{</td>
<td>statement;</td>
</tr>
<tr>
<td>statement;</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

- For this course, we insist you always use curly braces even when there is only one statement inside.
Complex if-else

• When you nest two or more if statements together:
  
  ```java
  if (expression1)
    if (expression2)
      if (expression3)
        statement1
      else
        statement2
  else
  ```

• The rule is that the last else is associated with the closest previous if statement that does not have an else component.
Avoid dangling \textit{else}

- To force the else to be associated differently, use \{ \} braces:
  
  \begin{verbatim}
  if (expression1){
    if (expression2){
      if (expression3){
        statement1
      }
    }
  }else {
    statement2
  }
  
  \end{verbatim}

- It is good programming style to always include braces, for clarity.
The else-if

- To create a multi-way decision chain:
  ```java
  if (condition_1) {
    statements_1;
  }
  else if (condition_2) {
    statements_2;
  }
  ...
  else if (condition_{n-1}) {
    statements_{n-1};
  }
  else {
    statements_n;
  }
  ```

- Evaluates conditions until finds a True one
- Then executes corresponding statements.
- Then finishes if statement

Phạm Bảo Sơn
If example: Dating for CS

```c
int age;
printf("How old are you: ");
scanf("%i ", &age);
if (age < 18) {
    printf("Do you have an older sister/brother? ");
} else if (age < 25) {
    printf("Doing anything tonight? ");
} else if (age < 35) {
    printf("Do you have a younger sister/brother? ");
} else if (age < 65) {
    printf("Do you have a daughter/son? ");
} else {
    printf("Do you have a granddaughter/grandson? ");
}
```
Conditional Expression

- Conditional expressions have the form:
  \[ \text{expr1? expr2 : expr3} \]

- Typical usage:

  ```
  if (x < a){
    z = x;
  } else{
    z = a;
  }
  ```

  Equivalent to:

  ```
  z = (x < a)? x : a;
  ```

- Because it is an expression, it can be used whenever any expression are used. Use with caution!

- You are advised to parenthesize `expr1` because of precedence.
Single or double equals

• Note the difference between \( = \) and \( == \)
  \( x = y; \) // store the value of \( y \) into \( x \)
  if (\( x == y \))… // check if values of \( x, y \) are equal

• In C an assignment evaluates to the value assigned:
  – if (\( a = 10 \)) … is always true
  – if (\( a = 0 \)) … is always false
  – if (\( a = b \)) … is equivalent to if ( (\( a=b \)) != 0) …
The switch statement

- Like the multi-way else-if statement, the switch statement behaves in a similar manner:

```
switch(expression) {
    case const-expr:
        statements
    case const-expr:
        statements
    default:
        statements
}
```
The switch statement (cont.)

- Each *case* must be a constant integer and not an expression.
- The *default* is optional.
- If a case matches the expression value, the execution starts at that case.
- If none of the cases match, then the default action is executed.
- If there is no default and no cases match, then no action takes place.
- The case and default can occur in any order (but only one default is allowed per switch statement)
The switch statement (cont.)

- *break* is used to force an immediate exit from the *switch* statement upon a case *const-expr* match.
- If *break* is omitted, then execution will flow on into the next case label, this is called “*falling though*” from one case to another.
- It is good practice to put a break at the end of the default even it it not necessary.
- *Fall through* code is not considered a good practice and should be avoided where possible. If it cannot, then make sure you flag this in your comments and make it very obvious.
Example of switch

```java
switch (month) {
    case 2:
        length = (year%4 == 0 &&
                    (year%100!=0 || year%400==0))? 29: 28;
        break;
    case 4: case 6: case 9: case 11:
        length = 30;
        break;
    default:
        length = 31;
        break;
}
```
Repetition

• C has several control structures for repetition:
  – while: zero or more times
  – do .. while: one or more times
  – for: zero or more time with initialization and update.
Repetition

• All repetition structures control:
  – A single statement or
  – A block of statements in {…}

• Repetition statements are also called loops.

• The control statement(s) are called the loop body.
Cấu trúc lặp

block

bool expr

block

bool expr
The while statement

- Repetition is controlled by a continuation condition, tested before the loop body is executed. Its general form is:
  
  ```
  while (condition){
      statement
  }
  ```

- Effect:
  - Test the continuation condition
  - If FALSE, end the while statement
  - If TRUE, execute the statements
  - Repeat the above three steps.
while example

• Compute the sum of the first 50 positive integers:

```c
int sum, num;
sum = 0;
num = 1;
while (num <= 50){
    sum = sum + num;
    num = num + 1;
}
```
The do while statement

- Repetition is controlled by a continuation condition, tested after the loop body is executed. Its general form is:
  ```java
  do {
      statement
  } while (condition);
  ```
- Effect:
  - Execute the statements
  - Test the continuation condition
  - If FALSE, end the do..while statement
  - If TRUE, repeat the above three steps.
The for statement

- The for statement is shorthand for a common pattern of usage of while:

  ```
  init; while (condition){
    statements;
    next;
  }
  ```

  ```
  for (init; condition; next){
    statements;
  }
  ```

- `init` sets state for first iteration, `next` sets state for next iteration.
- Any of `init`, `condition`, or `next` may be omitted.
- `for` is normally used for a fixed number of iterations.
Example of for

```c
int n, i, factorial;
printf("n = ");
scanf("%i ", &n);
for (i = 1, factorial = 1; i <= n; i++){
    factorial = factorial * i;
}
printf("%d ! = %d\n", n, factorial);
```
break and continue

- break causes a loop to terminate; no more iterations are performed, and execution moves to whatever comes after the loop.
- continue causes the current iteration of the loop to terminate; execution moves to the next iteration:
  - Note the difference between for loop and while/do-while.
- Avoid using break and continue in this course
References

• [K&R] Chapter 3.