Fundamentals of C++ Programming

Control Structures (part 2)
Lecturer: Duc Dung Nguyen
Credits: 4
Outcomes

❖ Using array, string, and structured data types
❖ Solve the problem using loop structures
❖ Implement program with loop structures:
   ❖ while, for, do-while
❖ Understand the role of algorithm in problem solving process
Today’s outline

❖ Structured data types
  ❖ Array
  ❖ Struct
❖ Basic control structures in C/C++
  ❖ Loop statements: while, for, do-while
❖ Structure programming
Structured data types
Structured data types

❖ Can we implement a program with only basic data types?
❖ What do we need beside basic data types?
   ❖ A sequence of memory slots that contains a specific data type
   ❖ A mixture of different data types
Structured data types

✦ Array: a sequence of memory slots that contains a specific data type

✦ `<data type> <variable name>[<Size>]`;

✦ `int Fibonacci[MAX_LENGTH];` /* declare an integer array of MAX_LENGTH elements. This is a static declaration! */

✦ `<data type> *<variable name>`; // alternative declaration, a pointer

✦ `float *plotY;`
Structured data types

❖ Array

❖ int N;
   cout << "Please input size of the sequence: ";
   cin >> N;
   float x[N]; // compiler will fire an error here
   ...

❖ int N;
   cout << "Please input size of the sequence: ";
   cin >> N;
   std::vector<float> x(N); // no error, but x is a vector class
   ...

Structured data types

❖ Array: initialization

❖ At declaration time (static)
  ❖ int sNum[5] = {5, 6, 9, 2, 1};
  ❖ float x[] = {0.1, 3.2, 5.7, 7.2}; // allocated 4 elements

❖ Dynamically allocate
  ❖ float *pNum;
    ...
    pNum = new float[N];
Structured data types

❖ Array

❖ Access array elements:

❖ `<variable name> [<index>]`

❖ int sNum[5];
  sNum[0] = 1;
  sNum[1] = 1;
  sNum[2] = sNum[0] + sNum[1];
Structured data types

❖ String:

❖ char strName[50]; // undefined string
❖ char strName[50] = “Dustin”;
❖ char strOutText[] = “This text contains 32 characters”; // 33 bytes
❖ char *pStr = “Unknown”;
Structured data types

❖ String:
  ❖ `strlen`: length of string
  ❖ `strcpy`: copy a string
  ❖ `strcat`: concatenate strings
  ❖ `strcmp`: compare two strings
  ❖ `strchr`: locate the first occurrence of character in a string
  ❖ `strrchr`: locate the last occurrence of character in a string
Structured data types

❖ **String:**

❖ `string sText; // empty string ""`

❖ `string sText = "A C++ class that stores characters.";`

❖ `sText.size, sText.length, sText.max_size, sText.resize, etc.`

❖ `sText[index], sText.at(index)`

❖ `sText += anotherText;`

❖ `sText.push_back, sText.pop_back, sText.find, sText.copy, etc.`
Structured data types

❖ Struct:

❖  `struct [<struct name>] {`
  
  `<elements>;
  
} [<variables>];`

❖  `struct Student {`
  
  `int ID;`
  
  `char name[50];// can you use string? Why should you use?`

  `};`

  `struct Student studentList[40];`
Structured data types

- **typedef**: define a data type
  
  ```c
  typedef struct { char name[30]; } StdName_t;
  StdName studentList[50];
  ```

  ```c
  typedef struct Student {
    int ID;
    char name[50]; // can you use string? Why should you use?
  } Student_t;
  Student_t studentList;
  ```
Control structures
while statement

- Why do we need iterations?
  - Waiting for something to happen
  - Operate on several objects
    - List, array of objects
    - String
while statement

❖ while loop:

❖ Execute a section of code over and over under certain conditions

❖ while (<condition>) <statement>;

❖ while (<condition>) {
   <statements>;
}

❖ E.g.:

❖ while (i < nItems) { checkItem(i); i++; }
while statement

- Flowchart
while statement

❖ Breaking the rule

❖ while (<condition>) {
    <statements>;
    if (<special condition>) break;
    <statements>;
}

CuuDuongThanCong.com
https://fb.com/tailieudientucntt
while statement

❖ Breaking the rule

❖ while (<condition>) {
    <statements>;
    if (<special condition>) continue;
    <statements>;
}

CuuDuongThanCong.com
https://fb.com/tailieudientucntt
while statement

❖ Breaking the rule

❖ while (<condition>) {
    <statements>;
    myLabel:
    <statements>;
    if (<special condition 1>) goto myLabel;
    if (<special condition 2>) goto exitLabel;
    <statements>;
}

exitLabel:
<statements>
while statement

❖ Note:

❖ Remember to initialize variables in the condition expression before entering the `while` statement (at least you know what will happen when you check the condition).

❖ Do not forget stopping condition.

❖ Take care of counters.

❖ Use infinite loop wisely.

CuuDuongThanCong.com

https://fb.com/tailieudientucntt
Nested loop

A loop can be nested inside a loop.

```
while (<condition 1>) {
    <statements>;
    while (<condition 2>) {
        <statements>;
        while (<condition 3>);
    }
    <statements>;
}
```
while statement
while statement

❖ Nested loop
  ❖ Is used to process multi-dimension arrays
  ❖ Access customised data
  ❖ Waiting for inputs
  ❖ etc.
Why do you need for statement?

- Just another way to write iteration/loop structure!
- Counting is a frequent activity
- for: a specialised loop that package the following tasks in a statement
  - Initialise a counter variable
  - Modify the counter
  - Check complete condition
for statement

❖ for loop:

❖ for (<initialization>; <condition>; <modification>) <statement>;

❖ for (<initialization>; <condition>; <modification>) {
   <statements>;
}

❖ E.g.:

❖ for (i = 0; i < 100; ++i) cout << i << ", ";

❖ for (j = 0; j > -10; --j) cout << j << ", ";
for statement

❖ Flowchart
for statement

- Initialization: set value for the counter
  - Declare one or many counters (same type) and init them at once
  - Initialize many counters if needed
- Condition: a boolean expression that must be evaluated at each loop
- Modification: change value of the counter at each loop
for statement

- Nested loop:

```cpp
#include <iostream>
#include <math.h>

int main() {
    int img[12][16];
    for (int i = 0; i < 12; i++) {
        for (int j = 0; j < 16; j++) {
            img[i][j] = rand() % 256;
        }
    }
    return 0;
}
```

```cpp
#include <iostream>
#include <math.h>

int main() {
    int img[12][16];
    int i = 0, j;
    for (; i < 12; i++) {
        for (j = 0; j < 16; j++) {
            img[i][j] = rand() % 256;
        }
    }
    return 0;
}
```
for statement

❖ Breaking the rule

❖ for (<initialization>; <condition>; <modification>) {
    <statements>
    if (<special condition>) break;
    <statements>
}

❖ for (<initialization>; <condition>; <modification>) {
    <statements>
    if (<special condition>) continue;
    <statements>
}
do-while statement

- do-while: do first, check later
- A convenient way to perform some operations
- E.g.:
  - Asking user to input some values
  - Check if any input value was invalid
  - Loop to input again
do-while statement

❖ **do-while loop:**

❖ do <statement>; while (<condition>);

❖ do {
   <statements>;
} while (<condition>);

❖ E.g.:

❖ do { cout << "Please input a positive number" << endl;
   cin >> i; // should prompt user
} while (i < 0);
do-while statement

❖ Flowchart
do-while statement

❖ Breaking the rule

❖ do {
    <statements>;
    if (<special condition>) break;
    <statements>;
} while (<condition>);

❖ do {
    <statements>;
    if (<special condition>) continue;
    <statements>;
} while (<condition>);
do-while statement

❖ Nested loop

❖ do {
   <statements>;
   do {
      <statements>;
      while (<condition 3>) {
         <statements>;
      }
   } while (<condition 2>);
   <statements>;
} while (<condition 1>);
Structure programming
Structure programming

- Definition: a programming paradigm aimed at improving the clarity, quality and development time of a computer program by making extensive use of subroutines, block structures and for/while loops.

- Structured programming languages: ALGOL, Pascal, PL/I, Ada, C/C++, etc.
Problem solving

1. Define problem
2. Gather information
3. Explore
4. Plan
5. Act
6. Check
7. Generalize
8. Disseminate

Criteria vs. constraints
Known vs. unknown
Look at the problem from different viewpoints
Brainstorm
Evaluate against criteria
Troubleshooting
Make sense
Solution
Method
Apply to new situations

Don Woods & Philip Wankat

Don Woods & Philip Wankat

Don Woods & Philip Wankat
Structure programming

❖ Loop and array
  ❖ Loop is good for performing operations on arrays, strings.
  ❖ “while”, “do-while”, “for” are exchangeable.
  ❖ Fixed size data should be processed using finite loops.
Issues

❖ Infinite loops:

❖ while (100);
❖ while (true) cin >> i;
❖ do {
  } while (-20);
❖ for (;;);
❖ etc.
Issues

❖ Infinite loops:

❖ Sometimes your code is stuck in an infinite loop due to logic errors

❖ Factors: input, user interaction, or special computation

❖ E.g.:

❖ nInputs = 0;
   for (int i = 0; i <= nInputs; i++) {
      cin >> list[i];
      nInputs++;
   }
Issues

❖ Infinite loops:

❖ for (int i = 0; i < N; i++) {
  int j = 2;
  cout << i << " : ";
  while (j < i) {
    if (i % j == 0) {
      cout << j << " ";
      j++;
    }
  }
  cout << endl;
}
## Priority of operators

<table>
<thead>
<tr>
<th>#</th>
<th>Operator</th>
<th>group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>::</td>
<td>scope</td>
</tr>
<tr>
<td>2</td>
<td>++ -- ( ) . -&gt;</td>
<td>(postfix) unary</td>
</tr>
<tr>
<td>3</td>
<td>++ -- ~ ! + - &amp; * new delete sizeof (cast)</td>
<td>(prefix) unary</td>
</tr>
<tr>
<td>4</td>
<td>.* -.*.</td>
<td>pointer to member</td>
</tr>
<tr>
<td>5</td>
<td>* / %</td>
<td>arithmetic: scaling</td>
</tr>
<tr>
<td>6</td>
<td>+ -</td>
<td>arithmetic: addition</td>
</tr>
<tr>
<td>7</td>
<td>&gt;&gt; &lt;&lt;</td>
<td>bitwise shift</td>
</tr>
<tr>
<td>8</td>
<td>&lt; &gt; &lt;= &gt;=</td>
<td>relational</td>
</tr>
<tr>
<td>9</td>
<td>== !=</td>
<td>equality</td>
</tr>
<tr>
<td>10</td>
<td>&amp;</td>
<td>AND</td>
</tr>
<tr>
<td>11</td>
<td>^</td>
<td>XOR</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>&amp;&amp;</td>
<td>conjunction</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>*= /= %= += -= &gt;&gt;= &lt;&lt;= &amp;= ^=</td>
<td>= ?</td>
</tr>
<tr>
<td>16</td>
<td>,</td>
<td>sequencing</td>
</tr>
</tbody>
</table>
Problem solving - example

- Input and draw the following figure in terminal:
  - Input: N (number of lines)
  - Output: (in case N = 5)
    
    ```
    * 
    * * 
    * * * 
    * * * * 
    * * * * *
    ```
Problem solving - example

- Input and draw the following figure in terminal:
  - Input: N (number of lines)
  - Output: (in case N = 5)
    *
    **
    ***
    ****
    *********
Problem solving - example

❖ Input and draw the following figure in terminal:

❖ Input: N (number of lines)

❖ Output: (in case N = 5)

```
*       *
**     **
* *   * *
*  * *  *
*   *   *
```
Problem solving - example

❖ Parse strings

❖ The command line is given as follows:

❖ <program name> [<arguments>]

❖ E.g.: test -o log.txt -d

❖ Write a program that parse the command line and print list of arguments
Problem solving - example

❖ Math:
❖ Print prime numbers from 2 to N (N is the input value)
❖ Print Fibonacci sequence from 3 to N (N is the input value)
Summarise

❖ Array, string, and structured data types
❖ Understand loop structures: while, for, do-while
❖ The role of algorithm in problem solving process
❖ Using loop structures on arrays, strings
❖ Implements algorithms with loops
Quiz & homework