Fundamentals of C++ Programming

Basic components in C/C++ (part 2)

Lecturer: Dustin Nguyen
Outcomes

❖ Understand basic components of C++
❖ Using assignment operator
❖ How to format the output
❖ How to use libraries
❖ How to input values
❖ How to define macro, constants
Today’s outline

❖ Assignment operation
❖ Output format (console, string)
❖ Libraries functions
❖ Input with cin
❖ Macro definitions
Assignment operation
Assignment operation

❖ `<left operand> = <expression>`
❖ `return <left operand>`
❖ `<left operand>` can’t be constant
❖ Example:
  ❖ `pi = 3.1415;`
  ❖ `keyPressed = ‘q’;`
Assignment operation

❖ Assign at the declaration instruction:

❖ `int x = 10;`
❖ `int y{8};`
❖ `float z(10.01f);`
❖ `AnimalC monkey(10.5, 30);// use Class construction`
❖ `float 8.0 = f;`
Assignment operation

❖ What is the default type of constants?
❖ Can we assign different types of value to a variable?
Assignment operation

❖ Default type of constants depend on how you declare it
  ❖ 10: decimal value, default type depends on context
    ❖ 012: octal value
    ❖ 0x64: hexadecimal value
  ❖ 3.1415: default type is double
Assignment operation

- Suffixes to specify types:
  - u/U: unsigned. E.g.: 23u, 23U
  - l/L: long. E.g.: 75l, 75L, 75UL, 75ul, 75u, 75LU
  - ll/LL: long long. E.g.: 90l, 90LL, 90ull, 90ULL
  - f/F: float. E.g.: 3.1415, 6.02e23, 1.6e-19, 6.02e23f, 3.1415F
  - l/L: long double. E.g.: 3.1415L
- Q: is -23U a legal declaration?
Assignment operation

- NOTE: compiler does not warn you when you assign a value with different type to variable (default setting)
  - int a = 1024UL, c = 3.1415;
  - unsigned int b = -4096;
  - float x = 6.2830L;
- Compiler warns you when overflow problem occurs in the assignment op.
- Default cast operation will be applied without warning
Cast operator

- Implicit convert the value from one type to another type
- `<target type>`<expression>
- E.g.:
  - `int a = (int)3.9583;`
  - `float x = (float)a + 0.5f;`
  - `double y = (double)x * (double)a; // y = x * a; is fine`
Auto type

- *auto type appears from C++11 standard.*
- Should we use auto?
- Where can we use auto?
- auto type: good or bad?

Know what you are doing
Make you code clear
Debug
Output format
Output format

- Text output format
  - `printf("i = %d\n", i);`
  - `cout << "i = " << i << endl;`
- Using function is a convenient way to format output.
- Using I/O streams require a bit modification in the sequence.
Output format

- `printf(<format string>, arguments)`
  - Format string can contain format specifiers with the following syntax:
    - `%[flags][width][.precision][length]specifier`
    - specifier: `d/i, u, o, x/X`(uppercase), `f/F, e/E, g/G, a/A, c, s, p, n, %(escape character)`
    - flags: `+`, `-`, space, `#`, `0`
    - `.precision: .number, .`
    - `width: number, *`
# Output format

<table>
<thead>
<tr>
<th>specifier</th>
<th>output</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>d/i</td>
<td>signed decimal integer</td>
<td>-2354</td>
</tr>
<tr>
<td>u</td>
<td>unsigned decimal integer</td>
<td>3056</td>
</tr>
<tr>
<td>o</td>
<td>unsigned octal</td>
<td>342</td>
</tr>
<tr>
<td>x/X</td>
<td>unsigned hexadecimal integer</td>
<td>6f0c</td>
</tr>
<tr>
<td>f/F</td>
<td>decimal floating point[^1]</td>
<td>3.14159</td>
</tr>
<tr>
<td>e/E</td>
<td>scientific notation</td>
<td>3.14159e-05</td>
</tr>
<tr>
<td>g/G</td>
<td>use shortest representation</td>
<td>3.14159</td>
</tr>
<tr>
<td>a/A</td>
<td>hexadecimal floating point</td>
<td>-0xc.90dep-3</td>
</tr>
<tr>
<td>c</td>
<td>character</td>
<td>a</td>
</tr>
<tr>
<td>s</td>
<td>string</td>
<td>damn it</td>
</tr>
<tr>
<td>p</td>
<td>pointer address</td>
<td>b8000000</td>
</tr>
<tr>
<td>n</td>
<td>nothing will be printed, argument must be a pointer to a signed int. The number of printed characters are stored location pointed by the pointer.</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>print ‘%’ character</td>
<td>%</td>
</tr>
</tbody>
</table>
Output format

Examples

```c
#include <stdio.h>

int main() {
    printf("Characters: %c %c \n", 'a', 65);
    printf("Decimals: %d %ld\n", 1977, 650000L);
    printf("Preceding with blanks: %10d \n", 1977);
    printf("Preceding with zeros: %010d \n", 1977);
    printf("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
    printf("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
    printf("Width trick: %*d \n", 5, 10);
    printf("%s \n", "A string");
    // default cast operation is applied
    return 0;
}
```
Output format

❖ Using cout: require “iomanip” for formatting

❖ cout.width(<output width>): set width of the output result, both text and number

❖ cout << setw(<output width>)

❖ cout.precision(<number>): set maximum number of significant digits (set to 0 to reset this setting)

❖ cout << setprecision(<number>)

❖ cout << “*” << setw(4) << 8 << “*” << endl;
❖ cout << “*” << setprecision(4) << 12356.4 << “*” << endl;
Output format

❖ Save old settings:
  ❖ `ios::fmtflags old_settings = cout.flags();`
  ❖ `int old_precision = cout.precision();`

❖ Load settings:
  ❖ `cout.flags(new_settings);`
  ❖ `cout.precision(new_precision);`
Output format

❖ Fixed format for floating point number

❖ `cout.setf(ios::fixed, ios::floatfield);`
❖ `cout << fixed;`
❖ Reset to default: `cout.setf(0, ios::floatfield);`

❖ E.g.:

❖ `cout.setf(ios::fixed, ios::floatfield);
 cout.precision(2);
 cout << 3.14159 << ", " << 0.8642e-3;`
Library functions
Library functions

- Library is the place where you implement functions, classes to serve some specific tasks.
- Library contains:
  - Definitions: constants, macro, structure, class
  - Functions: implement specific algorithms, a unit of reusable code
  - Class implementations
Library functions

❖ Function: a named sequence of code that performs a specific task

❖ Definition

❖ <return type> <function name>(<in/out parameters>); // prototype
❖ <return type> <function name>(<in/out parameters>)
  {
    // your implementation
  }
Library functions

❖ Standard libraries:

❖ `<math.h>`, `<cmath>`
❖ `<string.h>`, `<cstring>`
❖ `<stdio.h>`, `<cstdio>`
❖ `<assert.h>`, `<cassert>`
❖ `<errno.h>`, `<cerrno>`
❖ `<time.h>`, `<ctime>`
Library functions

❖ Example

```cpp
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

int main() {
    string name;
    char   buffer[50];
    float  x;
    cout << "Please input your name:";
    gets(cin, buffer);
    name = buffer;
    cout << "Hello " << name << endl;
    cout << "Please input a real number:";
    cin >> x;
    cout << "square root of " << x << " is " << sqrtf(x) << endl;

    return 0;
}
```
Input with cin
Input with cin

- cin is object of class istream connected to the standard input stream oriented to narrow characters (type char)
- Defined in <iostream>
- Use ">>" operator to obtain the input from istream object (cin)
- E.g.:
  - float x; cin >> x;
  - int a, b; cin >> a >> b;
Input with cin

- The operator is overloaded for various types
  - Arithmetic types: int, float, unsigned, etc.
- Stream buffers
  - Extract as many characters as possible from the stream and put them to the stream buffer
- Manipulators: ios/ios_base, are used to format the input stream
  - ws, boolalpha/noboolalpha, skipws/noskipws, dec/hex/oct
Input with cin

Example

```cpp
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

int main() {
    char name[50];
    float x;
    int a;
    cout << "Please input your name:";    cin >> name;
    cout << "Hello " << name << endl;
    cout << "Please input a real number:";    cin >> x;
    cout << "square root of " << x << " is " << sqrtf(x) << endl;
    cout << "Please input hexadecimal value: ";
    cin >> hex >> a; // use manipulator std::hex
    cout << "The decimal equivalent is " << a << endl;

    return 0;
}
```
Input with cin

❖ String input:
  ❖ Spaces will be considered as stop condition
  ❖ Behaviour of continuously input operators are unpredictable with abnormal input from user
❖ `istream::get()`: equivalent to `getc()` in standard C
❖ `istream::getline()`: equivalent to `gets()` in standard C
# Example

```cpp
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

int main() {
    char   name[50];
    float  x;
    int    a;

    cout << "Please input your name:";
    cin.getline(name, 50);
    cout << "Hello " << name << endl;
    cout << "Please input a real number:"; cin >> x;
    cout << "square root of " << x << " is " << sqrtf(x) << endl;
    cout << "Please input hexadecimal value: ";
    cin >> hex >> a; // use manipulator std::hex
    cout << "The decimal equivalent is " << a << endl;

    return 0;
}
```
Macro definition
Macro definition

- `#define`/#`undef`: preprocessor directives
- Extend across single line of code
- No semicolon `;` at the end
- Use `\` to write the define instruction with multiple lines
Macro definition

❖ Define constants: \#define <identifier> <replacement>

❖ \#define MAX_LENGTH 50
❖ \#define MY_STRING "This is a constant string"
❖ \#define pi_2 3.14159/2
❖ \#define pi_2 1.570785

❖ Constants variables:
❖ const float x_2 = x / 2; // x_2 cannot be changed
Macro definition

❖ Macros:

❖ `#define sub(a, b) a - b`
❖ `#define sub(a, b) (a - b)`
❖ `#define sub(a, b) ((a) - (b))`
❖ `#define swap(a, b, c) {
  c = a;
  a = b;
  b = c;
}
`
Macro definition

❖ More examples:

❖ #define NORMALIZE_FACTOR  50
   ...
   float fx = sum / NORMALIZE_FACTOR;
❖ #define sub(a, b) ...
   ...
   float x = 0.5 * sub(z + 3.9, y + f(t));
Macro definition

❖ Special operators:

❖ #: create a string literal that contains the argument

❖ #define text(a) #a

...  
cout << text(Be careful) << endl; // print “Be careful”

❖ ##: concatenate two arguments

❖ #define glue(a, b) a ## b

...  
glue(c, out) << “Weird way to write code\n”; // but acceptable
Summarise

❖ Understand basic elements of C/C++
  ❖ Assignment operator, default types, type casting, overflow problem
  ❖ Format the output
  ❖ Use library functions
  ❖ Input values
  ❖ Macro, constants
Quiz & homework

❖ Describe ways to assign value to a variable.

❖ When we print the output text, which format method is better? Describe your explanation supported by examples.

❖ Check functions in `<cmath>`, `<cstdio>`, `<cstring>`, `<cassert>`. Write code to test these functions. Report in class if you encountered any problem.