CS11600: Introduction to Computer Programming (C++)

Lecture 4

Svetlozar Nestorov
University of Chicago

Outline

- Introduction to C and C++
- “Hello World!” program
- Basic types
- Variables, constants and assignments
- Expressions and operators
- Control flow
- Input/Output (I/O)

Introduction to C and C++

- The history of C and C++
  - http://cm.bell-labs.com/cm/cs/who/dmr/chist.html
  - C was devised in 70’s at Bell Labs
  - Predecessors of C: BCPL, B.
  - Standardization: ANSI C in 1989.
  - C++ is the most widely used successor of C.
- Differences between C and C++

“Hello World” Programs

- C program

```
#include <stdio.h>
int main() {
  int year = 2003;
  printf("Hello %d World!\n", year);
}
```

- C++ program

```
#include <iostream.h>
int main() {
  int year = 2003;
  cout << “Hello ” << year << “ World!” << endl;
}
```

Built-in Types

- Integers (signed or unsigned):
  `int, short, long`
- Real numbers (always signed):
  `float, double, long double`
- Characters (signed or unsigned):
  `char`
- Others:
  `wchar_t, bool`

Variables

- Variable definition:
  `Type name [= value];`
- Examples:

```
int year = 2003;
int year;
char c = ‘c’;
double pi = 3.14;
```

- Note: variable name must start with a letter or _ and cannot be a C++ *keyword.*

More Variable Definitions

- Alternative initialization syntax:

```
 Type name(value);
```

- Multiple variable definitions:

```
Type name1(val1), name2(val2)…;
```

- Examples:

```
int year = 2003, nextYear;
char c(‘c’), d=‘d’, e;
```

Constants

- Constants (aka constant variables):

```
 const Type name = value;
```

- Examples:

```
const int year = 2003;
const char c = ‘c’;
const char c2 = c;
```

- Note: constant must be initialized and this initial value cannot be changed.

Assignments

- Basic form:

```
name = expr;
```

- `expr` can be a value (literal constant), constant, variable, or an expression.
- Another form:

```
 name1 = name2 = name3 = expr;
```

- Assigns `expr` to all three variables.

Basic Math Operators

- Arithmetic

```
-, +, -, *, /, %
```

- Logical

```
! (not), && (and), || (or)
```

- Bitwise

```
~ (compl), & (bitand), | (bitor), ^ (xor)
<<, >>
```

- Relational (comparisons)

```
<, >, <=, >=, ==, != (not_eq)
```

Update Operators

- Update operators:

```
+=, -=, *=, /=, %=, <<=, >>=, &=, |=, ^=
```

- General form:

```
name update_op expr;
```

- Equivalent to:

```
name = name op expr;
```

- Examples:

```
i += 5; (same as i = i + 5;)
c *= 10; (same as c = c * 10;)
```

Increment and Decrement

- There are 4 variations:
  - Prefix increment: `++name`
  - Prefix decrement: `--name`
  - Postfix increment: `name++`
  - Postfix decrement: `name--`
- Prefix updates the value *before* it is used in an expression while postfix updates it *after* it is used.
- Work for numbers and characters.

More Operators

- Conditional operator

```
cond ? expr1 : expr2
```

- Evaluates to `expr1` if `cond` is true, otherwise evaluates to `expr2`
- Comma operator

```
expr1, expr2, …
```

- Evaluates expressions left to right; value of operator is the last expression.

Control Flow

- Three kinds of control flow constructs:
  - Conditional: `if, if-else, switch`
  - Loop: `for, while, while-do`
  - Jump: `break, continue, goto`

If

- If has three forms:

```
if (condition) {
  statements;
}
```

```
if (condition) {
  statements1;
}
else {
  statements2;
}
```

```
if (cond1) {
  statements1;
}
else if (cond2){
  statements2;
}
…
else {
  statements3;
}
```

Switch

- General form:

```
switch (expr) {
  case constant1:
    statements1;
    break;
  case constant2:
    statements2;
    break;
  …
  default:
    statements1;
    break;
}
```

- Default is optional; order of cases is arbitrary, several cases may share statements.

While

- Two forms:

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

- Evaluate condition, if true execute statements and repeat.

```
do {
  statements;
} while (condition)
```

- Execute statements, evaluate condition, if true repeat.

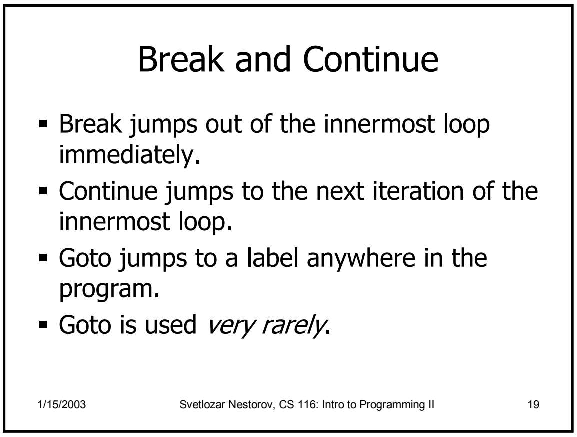
For

- General form:

```
for (expr1; cond; expr3) {
  statements;
}
```

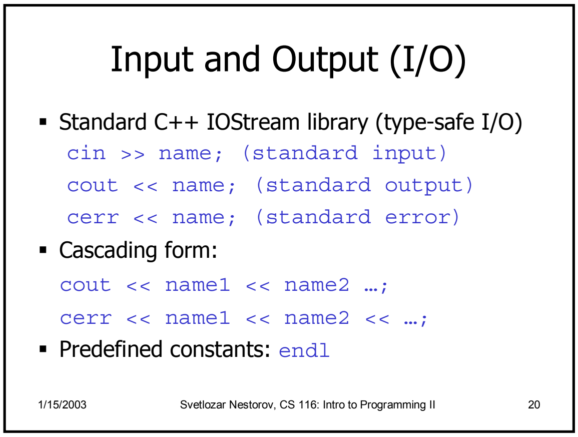
- Equivalent to:

```
expr1;
while (cond) {
  statements;
  expr3;
}
```



Break and Continue

- Break jumps out of the innermost loop immediately.
- Continue jumps to the next iteration of the innermost loop.
- Goto jumps to a label anywhere in the program.
- Goto is used *very rarely*.



Input and Output (I/O)

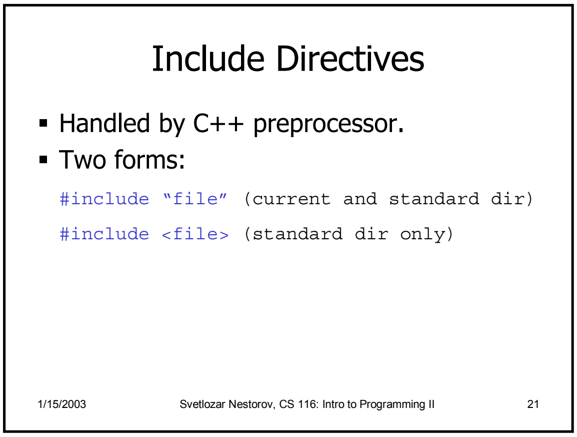
- Standard C++ IOStream library (type-safe I/O)

```
cin >> name; (standard input)
cout << name; (standard output)
cerr << name; (standard error)
```

- Cascading form:

```
cout << name1 << name2 …;
cerr << name1 << name2 << …;
```

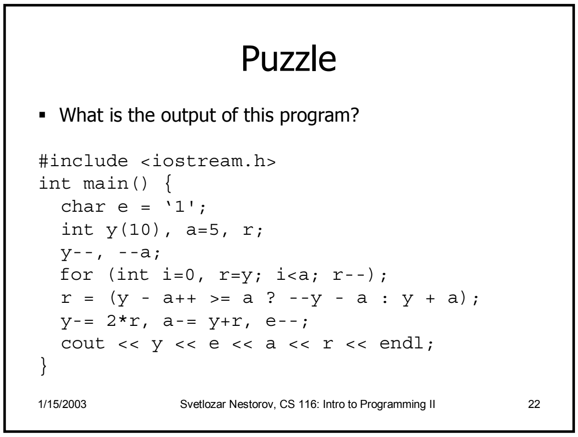
- Predefined constants: `endl`



Include Directives

- Handled by C++ preprocessor.
- Two forms:

```
#include “file” (current and standard dir)
#include <file> (standard dir only)
```



Puzzle

- What is the output of this program?

```
#include <iostream.h>
int main() {
  char e = ‘1';
  int y(10), a=5, r;
  y--, --a;
  for (int i=0, r=y; i<a; r--);
  r = (y - a++ >= a ? --y - a : y + a);
  y-= 2*r, a-= y+r, e--;
  cout << y << e << a << r << endl;
}
```