Object-oriented Programming

for Automation & Robotics

Carsten Gutwenger

LS 11 Algorithm Engineering

Lecture 7 • Winter 2011/12 • Nov 29

Arithmetic Type Conversions

- Sometimes we have to convert arithmetic data types, e.g. `int` to `double` or `float` to `long long`.
- Implicit Type Conversions
  - In arithmetic expressions of mixed types, the widest data type becomes the target type
  - In an assignment, the target type is the type of the object assigned to
  - In passing an expression as an argument to a function, the target type is the parameter type

```
int    ival = 3;
double dval = 3.14159;

// ival is promoted to double 3.0
ival + dval;  // expression
dval = ival;  // assignment
```

Explicit Type Conversions

- We get a compiler warning when a conversion to a “smaller” type is necessary:

```
int   ival = 3;
float fval = 3.14f;

fval = ival;
```

Compiler warning:
Conversion from ‘int’ to ‘double’; possible loss of data

- Explicit Type Conversion:
  - Forces type conversion to a specific type
  - General form:

```
static_cast<type-name>(expression);
```

forces ***expression*** to be converted to type ***type-name***

  - Example:

```
fval = static_cast<float>(ival);
```

The Comma Operator

- A comma expression is a series of expressions separated by commas
  - Expressions are evaluated from left to right
  - Result is the value of the rightmost expression
- Example:

```
void enter_pair(int &x, int &y) {
    cout << "x = "; cin >> x;
    cout << "y = "; cin >> y;
}

int main() {
    int x, y;
    while(enter_pair(x,y), x+y != 0)
        cout << "x + y = " << x+y << endl;

    return 0;
}
```

Function Overloading

- Allows to use the same name for multiple functions that provide a common operation on different parameter types
- Parameter lists must be unique in either the number or the types of parameters
- The compiler automatically chooses the “right” version of the function, depending on the argument types

Example: Overloading max functions

```
int max(int a, int b) {
    return (a >= b) ? a : b;
}

int max(int a, int b, int c) {
    return max(max(a,b),c);
}

int max(const vector<int> &v)
{
    if(v.empty()) return 0;

    int m = v[0];
    for(vector<int>::size_type i = 1; i < v.size(); ++i)
       if(v[i] > m) m = v[i];

    return m;
}
```

Example: main function

```
int main()
{
    vector<int> data;

    cout << "Enter positive numbers, stop with non-negative number: ";

    int x;
    while(cin >> x, x > 0)
       data.push_back(x);

    cout << "Maximum is: ";
    cout << max(data) << endl;

    cout << "max(5,40,20) = ";
    cout << max(5, 40, 20) << endl;

    return 0;
}
```

Example: Overloaded print functions

```
void print(int x)
{
    cout << "int:       " << x << endl;
}

void print(bool x)
{
    cout << "bool:      " << boolalpha << x << endl;
}

void print(unsigned x)
{
    cout << "unsigned:  " << x << endl;
}

void print(long long  x)
{
    cout << "long long: " << x << endl;
}
```

Example: main function

```
void print(int);
void print(bool);
void print(long long);
void print(unsigned int);

int main() {
    int       i = -5;
    long long l = 12345678901234567;
    bool      b = true;
    unsigned  u = 4;

    print(i);
    print(l);
    print(b);
    print(u);
    print(static_cast<unsigned>(i));
    print(static_cast<int>     (l));

    return 0;
}
```

Output:

```
int:       -5
long long: 12345678901234567
bool:      true
unsigned:  4
unsigned:  4294967291
int:       1567312775
```

Preparations for next week

- Custom types: **struct**
- Constructors
- Member functions (basics)
- Operator overloading