

CS 460

Programming Languages

Fall 2008

Dr. Watts

(Week 11)



Project 2 – sharing files

```
.....  
file.h  
.....  
#include <iostream>  
#include <fstream>  
using namespace std;  
extern ifstream input;  
extern ofstream output;  
void start ();  
void stop ();
```

```
.....  
file.cpp  
.....  
#include "file.h"  
using namespace std;  
ifstream input;  
ofstream output;  
void start ()  
{  
    input.open ("test.in");  
    output.open ("test.out");  
}  
void stop ()  
{  
    input.close ();  
    output.close ();  
}
```

```
.....  
app.cpp  
.....  
#include "file.h"  
  
using namespace std;  
  
int main ()  
{  
    cout << "opening files\n";  
    start ();  
    cout << "reading from input  
        file\n";  
    int i;  
    input >> i;  
    cout << "writing to output  
        file\n";  
    output << "the value is " <<  
        i << endl;  
    cout << "closing files\n";  
    stop ();  
    return 0;  
}
```





Record Types

- Collect of named data fields – generally heterogeneous
- Introduced by COBOL in the 1960s

```
01 EMPLOYEE_RECORD.  
  05 EMPLOYEE-NAME.  
    10 FIRST                PIC X (10).  
    10 MIDDLE-INIT         PIC X.  
    10 LAST                 PIC X (20).  
  05 SALARY                 PIC 999V99.
```

- Pascal and Ada record type
- C/C++ struct



Record Implementation

- Fields stored in adjacent memory locations
- Field accessed using offset from beginning of record
- May not be stored in order of declaration
- Operators
 - Comparison
 - Assignment
 - Input/Output

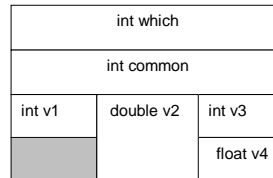
Union Types



- Variations on use of space
- C/C++

```
struct container
{
    int which;
    int common;
    union variable
    {
        int v1;
        double v2;
        struct
        {
            int v3;
            float v4;
        } v5;
    } u;
};
```

sizeof(c): 16 at 0x7fff65be5010
sizeof(c.which): 4 at 0x7fff65be5014
sizeof(c.common): 4 at 0x7fff65be5018
sizeof(c.u): 8 at 0x7fff65be5018
sizeof(c.u.v1): 4 at 0x7fff65be5018
sizeof(c.u.v2): 8 at 0x7fff65be5018
sizeof(c.u.v5): 8 at 0x7fff65be5018
sizeof(c.u.v5.v3): 4 at 0x7fff65be5018
sizeof(c.u.v5.v4): 4 at 0x7fff65be501c



Union Implementation



- Tag / discriminant
- Type checking
- Use the same base address for each element of the union
- Sufficient storage for largest element is allocated
- Not particularly safe – weak typing
- Predecessor of inheritance



Pointer Types

- Contains an address
- Special value to indicate empty – NULL, nil, null
- Space allocated at run time on heap
- Lifetime issues
 - Programmer controlled
 - System controlled
 - Garbage collection



Pointer Problems

- Dangling pointers

```
int * a;  
a = new int;  
*a = 5;  
delete a;  
cout << *a;
```

- Lost heap-dynamic variables (memory leak)

```
int * a;  
a = new int;  
a = new int;
```



Reference Types

- Contains address
- Syntactically treated as static variable

```
int variable = 0;
int & ref_variable = variable;
ref_variable = 100;
cout << variable << ' ' << ref_variable;
```

- 100 100
- C++ pass by reference
- C++ return by reference



Heap Management

- Allocation issues
- Single-size units vs. variable-size units
- Garbage collection
- Reference counters
 - Eager approach
 - Claimed as soon as available
- Mark and sweep
 - Lazy approach
 - Claimed when needed

Expressions and Assignment Statements



- Fundamental means of specifying computations in a programming language
- Issues
 - What operators are implemented?
 - What are the precedence rules?
 - What are the operator associativity rules?
 - What is the order of operand evaluation?
 - Are there restrictions on operation evaluation side effects?
 - Does the language allow operator overloading?
 - What type mixing is allowed in expressions?

What operators are implemented?



- Standard:
 - +, -, *, /
- Non-standard
 - Exponentiation: **, ^
 - Modulo: %, rem, mod
 - Increment and decrement: ++, --
 - Bit shift: <<, >>
 - Logical: <, .LT., or <>, !=, .NE.
- Assignment?
 - = (C++) vs := (Pascal)

What are the precedence rules?



- C++
 - Group 1: ::
 - Group 2: () [] -> . ++ --
 - Group 3: ! ~ ++ -- + * &
 - Group 4: ->* .*
 - Group 5: * / %
 - Group 6: + -
 - Group 7: << >>
 - Group 8: < <= > >=
 - Group 9: == !=
 - ...
 - Group 17: ,
- Ruby
 - Group 1: [] []=
 - Group 2: **
 - Group 3: ! ~ + -
 - Group 4: * / %
 - Group 5: + -
 - Group 6: << >>
 - Group 7: &
 - Group 8: |
 - Group 9: < <= > >=
 - ...
 - Group 15: = %= { /= -= +=
|= &= >>= <<= *= &&= ||=
**=

What are the operator associativity rules?



- Left to right
 - $A - B + C$
 - $A / B * C$
- Right to left
 - $A -= B += C$
- Does it matter?
 - $(A - B) + C$ vs. $A - (B + C)$
- Overflow issues

What is the order of operand evaluation?



```
#include <iostream>
using namespace std;

int g;

int f (int p)
{
    g += p;
    return g;
}

int main ()
{
    g = 5;
    cout << g << ' ';
    cout << f(3) << endl;
    g = 5;
    cout << g << ' '
        << f(3) << endl;
    return 0;
}
```

- What is the output?

What is the order of operand evaluation?



```
#include <iostream>
using namespace std;

int g;

int f (int p)
{
    g += p;
    return g;
}

int main ()
{
    g = 5;
    cout << g << ' ';
    cout << f(3) << endl;
    g = 5;
    cout << g << ' '
        << f(3) << endl;
    return 0;
}
```

- What is the output?

5 8

8 8

- Why?



Conditional Operator

- C/C++ and descendants

```
a = b < c ? 2 * b : c - 2;
```

- is

```
if (b < c)
    a = 2 * b;
else
    a = c - 2;
```

- Flexibility

```
for (a = b < c ? 2 * b : c - 2; a < b; a++, b++)
```



Are there restrictions on operation evaluation side effects?

- $x + f(y)$ vs. $f(y) + x$

- Same if $f(y)$ has no side effects that modify x

- $(a + f(x)) * (b + f(x))$

vs.

- `Temp = f(x)`

```
(a + temp) * (b + temp)
```

- Only same if $f(x)$ has **no** side effects

Does the language allow operator overloading?



- C++
 - Operators can be overloaded for aggregate types
- Java
 - Operators cannot be overloaded by user
 - + overloaded for String type
- Ruby
 - Operators can be overloaded for all types – including literals and “primitives”
- Benefits
 - Consistency of syntax
- Drawbacks
 - Unexpected results

What type mixing is allowed in expressions?



- C++ primitive types
 - Type mixing in expressions results in type coercion
 - Widens to greatest magnitude/precision
- C++ user defined types
 - Expressions can mix types only if explicitly defined

C/C++ Implicit type coercion



	char	int	long	float	double
char	char	int	long	float	double
int	int	int	long	float	double
long	long	long	long	float	double
float	float	float	float	float	double
double	double	double	double	double	double

Errors in Expressions



- Compile time
 - Type conversion errors
- Run time
 - Overflow
 - Underflow
 - Floating point exceptions
- Exception handling



Relational Expressions

- Relational operators
 - <, <=, >, >=, == or =, != or /= or <>
 - Result types
 - Boolean
 - Integer
- Boolean Operators
 - And (&&), Or (||), Xor (^), Not (!)
 - Precedence
 - !; == and !=; && ||



Short Circuit Evaluation

- Used in evaluation of boolean statements
 - `if (cond1 and cond2)`
 - `cond2` is evaluated only if `cond1` is true
 - `if (ptr != NULL && ptr->field > 0)`
 - `if (cond1 || cond2)`
 - `cond2` is evaluated only if `cond1` is false
- Language dependent
 - C/C++
 - && and || are evaluated using short circuiting
 - & and | do not use short circuiting
 - Ada
 - Can be specified: “and” vs. “and then”
 - Perl, Ruby, Python
 - && and || short circuited



Assignment statements

- Simple assignment
 - `<variable> <assignment op> <expression>`
`a = b + c; V[i] = c - d;`
`a := b + c; V[i] := c - d;`
- Compound assignment
`a += b; c -= d + e;`
- Assignment chaining
`a = b = c + d;`



Assignment statements

- Conditional targets
 - `condition ? variable1 : variable2 = expression`
 - Example
`flag ? count1 : count2 = 0`
 - Equivalent to
`if (flag)`
 `count1 = 0;`
`else`
 `count2 = 0`



Assignment statements

- Assignments as expressions

- Examples

```
if (a = b + c)
```

```
x = (a = b + c) * (d = e - f)
```

- Book example

```
a = b + (c = d / b++) - 1
```

What does this do?



Assignment statements

- Assignments as expressions

- Examples

```
if (a = b + c)
```

```
x = (a = b + c) * (d = e - f)
```

- Book example

```
a = b + (c = d / b++) - 1
```

```
    c = d / b
```

```
    b++
```

```
    t1 = b + c
```

```
    a = t1 - 1
```

Expression evaluation implementation



- Precedence grammar
- Stack based evaluation
 - Examples:
 - $a = 5 + 2 * 3 - 7 / 4$
 - $b = (a + c * 2) - (c + (d - 2 * e))$
 - Where $c = 3$, $d = 4$, and $e = 5$