SFU CMPT-212 2008-1 Topic: Basic Types

Ján Maňuch

E-mail: jmanuch@sfu.ca

Friday 11th January, 2008

Types

Recall: each variable has declared — declaration is done by specifying the type.

Two categories:

- *base types:* integer types and floating-point types
- *compound types:* arrays, structures, pointers

Integer types

- `bool` (8 bits, values `false` (0) and `true` (1))
- `char` (8 bits)
- `short` (at least 16 bits)
- `int` (dependent on architecture (machine), between short and long)
- `long` (at least 32 bits)

Remarks:

- `unsigned` and `signed` can be added to the all integer types (except bool), extending (doubling) its positive coverage.
- `short`, `int` and `long` are *signed* by default, `char` can be either (depends on compiler)
- `long` can be added to some types, increasing (doubling) their size (not applicable on older compilers).

sizeof

- The operator `sizeof` can be used to determine the size in bytes of any type or variable. The value returned is the number of bytes used when the system allocates memory for a variable of the given type resp. for the variable.

  Examples: sizeof.cpp, sizeof2.cpp

- `climits` header file

  Examples: slimits.cpp, ulimits.cpp

- `limits` header file (uses templates)

  Example: new_limits.cpp

Sample questions

- List all integer types.
- Why is it recommended to use `int` type as integer type whenever possible?
- What would the following statement print?

```
1 cout <<sizeof(long long)-sizeof(unsigned long long);
```

The char type

- a small integer designed for storing characters
- `cin` and `cout` treat `char` variables in different way: chartype.cpp
- encoding between characters and characters codes is most likely ASCII (can be EBCDIC too)
- char constants: `'M'`, `'\n'`
- *wide character type* `wchar_t` — to represent extended set of characters, see page 86 in [Prata]

Q: are the following two C++ statements equivalent?

```
1 char initial=77;
2 char initial='M';
```

Floating-point types

- `float` (at least 32 bits)
- `double` (at least 48 bits)
- `long double` (at least as many bits as `double`), see sizeof.cpp
- types differ in the number of *significant digits* of mantissa and the *minimum* and *maximum exponent* values

`2.578E7`

`2.578` — mantissa

`7` — exponent

`2.578E7`$= 2.578 \times 10^7 = 25,780,000$

Examples: float.cpp new_float.cpp

Constants

- The `const` keyword introduces declarations of constants, identical with regular variables, but whose value cannot be changed.

```
1 const unsigned int points=5;
2 // constant declaration has to be initialized
3 points=10; // compiler error
```

- **Explicit Constants:** Represent values that are explicitly written inside the program. The compiler automatically allocates them a type.

Examples:

- `int`: `1`, `-2000`, `0x37` (hexadecimal), `037` (octal)
- `unsigned int`: `1u`, `12U`
- `long` and `unsigned long`: `1L`, `-3L`, `100UL`, `-5678lu`
- `300000000000` — compiler chooses the smallest type (starting from `int`) which can represent the constant
- `char`: `'A'`, `'\n'`, `'\101'` (octal), `'\x41'` (hexadecimal) some other special `char` constants see page 83 in [Prata]
- `float`: `1f`, `1.023e24F`
- `double`: `1.0`, `.12`, `2.11e-34`
- `long double`: `3.1415926L`, `1E4000l`
- character strings: `"hello C++"`, `"ab\x1ac"`

Basic Numeric Operators

- Assignment: `=`
- Arithmetic: addition (`+`); subtraction (`-`); product (`*`);
  division (`/`); modulus (`%`);
  unary increment (`++`); unary decrement (`--`)
- Logic (for `bool` type): not (`!`); or (`||`); and (`&&`)
- Bitwise: bit-or (`|`); bit-and (`&`); bit-xor (`^`);
  bit left-shift (`<<`); bit right-shift(`>>`)
- (precedence and associativity, see Appendix D of [Prata] or
  `http://web.ics.purdue.edu/~cs240/misc/operators.html`)

Comments:

- *division* acts in different ways depending on type:

```
1 cout <<10/3;     // prints 3
2 cout <<10.0/3.0 // prints 3.33333
```

- *modulus* is a counterpart to integer division

```
1 cout <<10%3;     // prints 1
```

suitable for testing divisibility (e.g., odd or even)

- for arithmetic operations precedence is natural

```
1 cout <<2+3*2;   // prints 8 (not 10=(2+3)*2)
2 cout <<10/5*2;  // prints 4 (not 1=10/(5*2))
3 // *, / have left to right associativity
```

- unary increment and decrement operators can be both prefix and suffix

```
1 int a=4;
2 a--;          // decrements value of a to 3
3 --a;          // decrements value of a to 2
4 cout <<a++; // prints 2 and increments value of a
5 cout <<++a; // increments value of a and prints 4
```

- All basic binary operators can be combined with an assignment:

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

```
1 int a=5;
2 a += 2;  // increment value of a by 2
3 // equivalent to
4 a = a+2;
```

Sample questions

- Assume that `int` has only 2 bytes. What types the following constants have `30000`, `40000` and `80000`.
- What is the difference between `++a` and `a++`?
- Specify four different ways how to increment value of a variable `a`.

Type conversions

- when assigned a value of one arithmetic type to a variable of another arithmetic type
- expression with mixed types (or types smaller than `int`)
- when passing a wrong type value to a function

Phases of an assignment:

- conversion
- the actual assignment
- returns *assigned* value — this allows to chain assignments:

```
1 int a,b;
2 a=b=10; // the same as a=(b=10);
```

Note. assignment operators have right to left associativity

Comments:

- conversions in assignments are not always safe (loss of data, compiler usually gives warning)

```
1 int a=3.14;                 // a=3
2 unsigned short b=65537; // b=1
```

- conversions in expressions are safe: from smaller to bigger type

```
1 int a=10;
2 long b=20L;
3 cout <<a+b; // a is converted to long
4             // and then summed with b
```

- and are performed at least in `int`

```
1 short a=5, b=10;
2 short c=a+b;     // 3 conversions!
```

Type casting

"Tells" C++ to read a variable of a certain type as a variable of another type.

Syntax:

- C flavor: `(type) variable`
  Examples: `(int) myvariable; (int) 'C';`
- C++ flavor: `type(variable)`
  Examples: `int(myvariable); int('C');`
- newest C++ flavor: `static_cast<type>(variable)`
  Examples: `static_cast<int>('C');`

Conversions:

- `int('C')` will return the ASCII code for the character 'C'
- if `int x = 1;` then `(double) x` will return a variable of type `double` with the same numeric value as x:

```
1 int x=1;
2 cout <<(double) x; // prints 1.00000
```

Example: typecast.cpp [Prata]

Type casting and conversion

- Type casting and conversion are different operations. During conversion, the computer is providing a smart facility; during casting, the programmer is smarter than the computer, by overriding the compiler's type checking facilities.
- During type casting, the compiler gives no warnings, even if conversion is not safe (e.g., `double` to `int`).
- Many bugs can be avoided through the compiler's type checking facility. Circumventing it (using casting) can result in serious, hard to track errors.

Sample questions

- How many automatic conversions happen in the following code?

```
1 int i;
2 long l;
3 i=l=10;
```

- What will the following statement print?

```
1 char letter='A', one=1;
2 cout <<letter+one<<endl;
```

- List all of the automatic conversions which happen in the following code (in the order in which they happen):

```
1 short a=1,b=2;
2 int i;
3 double d;
4 i=d=a+b;
```

  For each conversion specify which type is converted to which type.

- Why you should avoid using type casting? Show one example when type casting can useful.