TAMPERE UNIVERSITY OF TECHNOLOGY

Many consts

4.12.2018

`const` – constants

- Immutable ”variables”
- Compile-time constants: `int array[MAX];`
- Normally inside a compilation unit
- Usually in header files

const – primitive types

```
int const STR_MAX = 30000;
double const PI = 3.14159265;
int const LIMIT = getLimit();
char str[STR_MAX];
```

Constant objects

- Objects can be made constants in the same way as variables:

```
Date const christmas(24,12,1999);
```

- What does a constant object mean?
- Assignment?

Constant objects

- The **state** of the object cannot be changed
- Two kinds of member functions: “changeable” functions & constant functions
- Two interfaces
- The specifier of constant member functions (**const**) after the parameter list

Date class

```
class Date {
public:
    Date(unsigned int d, unsigned int m,
unsigned int y);
    ~Date();

    void setDay(unsigned int day);
    void setMonth(unsigned int month);
    void setYear(unsigned int year);

    unsigned int getDay() const;
    unsigned int getMonth() const;
    unsigned int getYear() const;

    void proceed(int n);
    int howFarAhead(Date const& d) const;
```

```
private:
    unsigned int
day_;
    unsigned int
month_;
    unsigned int
year_;
```

Date class

```
void Date::setDay(unsigned int day)
{
    day_ = day;
}

unsigned int Date::getDay() const
{
    return day_;
}
```

Compiler checks for `const`

- Restrictions in the code of constant member functions
  - Member variables
  - Member functions
  - **`this`**

Compiler checks for `const`

- State of an object = member variables?
- State of an object $\subset$ member variables?
- State of an object $\supset$ member variables?

Constant references and pointers

- Constant objects are unusual, change is typical for objects
- Constant reference: `Date const& d;`
- Constant pointer: `char const* str;`
- (Pointer as a constant: `char* const str;`)

Constant references and pointers

- Object/variable referred/pointed by a reference/pointer behaves like a constant
- It is possible to restrict the interface of a non-constant object to a constant

Constant references and pointers

- Documentation means
- Protection mechanism

```
void changeAnyway(Date const& d) {
   d.setDay(1); // COMPILATION ERROR:
                // setDay not a constant member
function
}
```

Conversion to constants but not vice versa

- C++ has implicit type conversion: normal pointer → constant pointer
- C++ has **not** implicit type conversion: constant pointer → normal pointer

```
char* str = "Use string class instead of char*";
char const* const_str = str;  // Ok: non-constant → constant
char* str2 = const_str;       // COMPILATION ERROR: constant →
non-constant
```

The importance of `const` specifier

- `Added compiler checks`
- Documentation means
- Constant objects can be pointed/referenced only by constant pointers/references
- One missing **`const`** → using the others becomes problematic

Error chain with const

```
#include “date.hh"

bool hasLeapDay(Date* date_p);

bool affordsSkirt(Date const& now)
{ // A month has a leap day if the month is February and the year
is a leap year
    if (now.getMonth() != 2) {
        return false; // Not February
    }
    else {
        return hasLeapDay(&now); // COMPILATION ERROR
    }
}
```