Programmering i C++
EDA623
Exceptionella händelser

Exceptionella händelser

Innehåll

- Att generera exceptionella händelser
- Att fånga exceptionella händelser
- Specifikation av exceptionella händelser

Exceptionella händelser

- Hantering av olika typer av fel (t.ex. indexering utanför gränser) kan göras med `throw` och `catch`
- Vidarebefordran av felhantering: `throw`
- Uppfångning av felhantering: `catch`
- Exceptionella händelser kan också benämnas *undantag*
- Kräver direktivet

```
#include <stdexcept>
```

Att generera exceptionella händelser

Syntax för throw

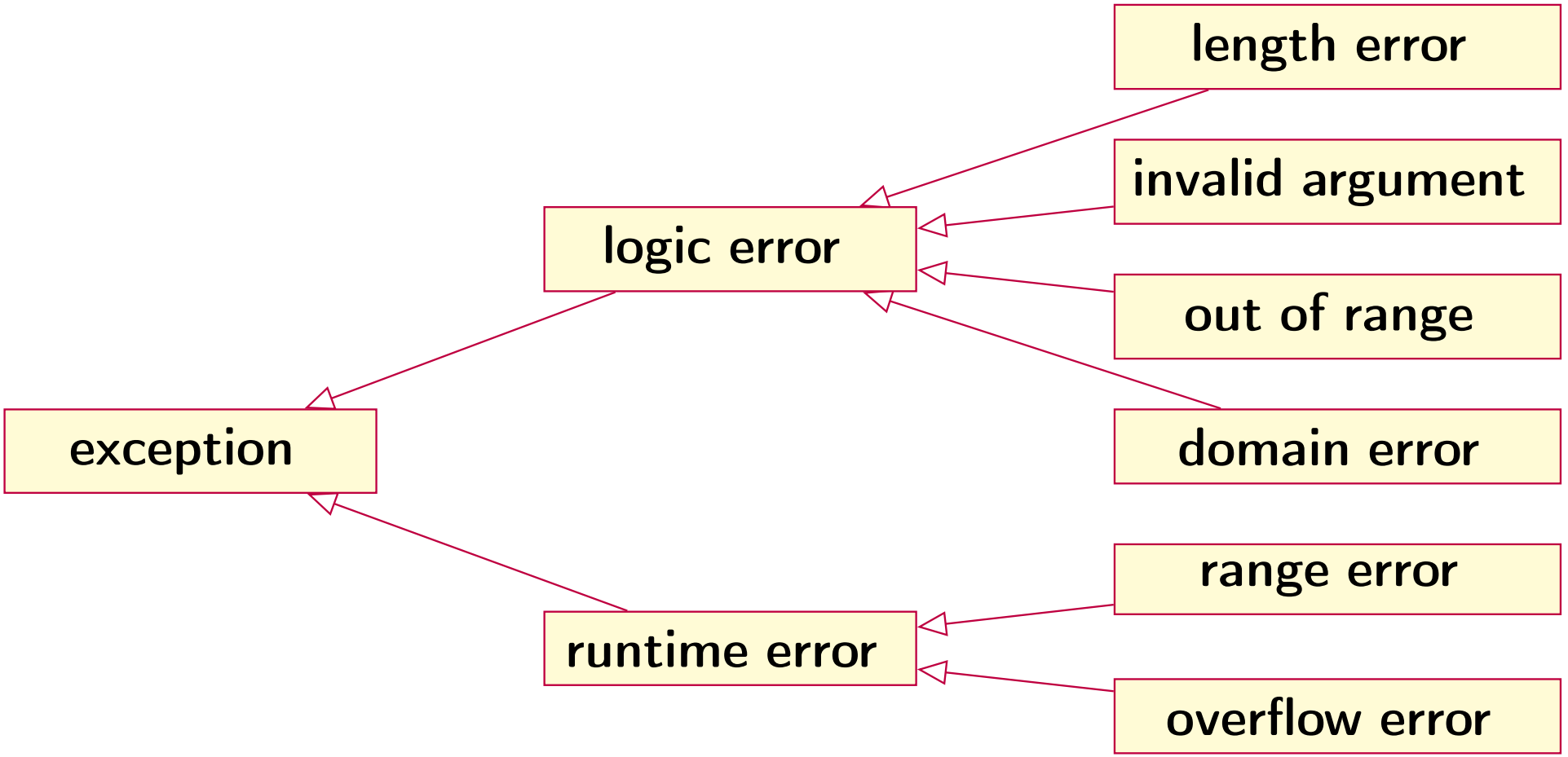
```
throw exceptionnamn("Extra info");
```

Exempel:

```
throw invalid_argument("till f2");
```

Att generera exceptionella händelser

Klassträd för `exception`



Att generera exceptionella händelser

Exempel: Användning i klassen `Vektor`

Ersätt alla assert med `throw`-satser istället:

```
int& Vektor::operator[] (int i) {
    if (i<0 || i>=ant)
        throw out_of_range("Vektor::operator[]");
return p[i];
}

const Vektor& Vektor::operator+= (const Vektor& v) {
    if (ant != v.ant)
        throw length_error("Vektor::operator+=");
    for (int i=0; i<ant; i++) {
        p[i] += v.p[i];
    return *this;
}
```

Att generera exceptionella händelser

Deklaration av egna undantag som subklasser

```
class communication_error : public runtime_error {
public:
    communication_error(const string& mess = "")
        : runtime_error(mess) {}
};
```

Användning av egendeklararade undantag

```
throw communication_error("Checksum error");
```

Att fånga exceptionella händelser

Tre nivåer av felhantering:

1. Vidta lämplig åtgärd direkt för att möjliggöra fortsatt exekvering
2. Kategorisera och skicka vidare felet till någon annan programenhet, vilken förväntas hantera det
3. Identifiera felet, ge något felmeddelande samt låt därefter programmet krascha

Att fånga exceptionella händelser

Undantagshanteringen är inriktad på fall 2: Felet skickas vidare med throw och fångas sen upp i en annan programdel med try-catch:

```
try {
    // Programkod där fel kan uppstå
}
catch (parameter av någon undantagstyp) {
    // Kod som tar hand om denna typ av fel
}
catch (parameter av någon undantagstyp) {
    // Kod som tar hand om denna typ av fel
}
catch (...) {
    // Kod som tar hand om resten (default)
}
```

Att fånga exceptionella händelser

Exempel:

```
int i;
try {
    cout << "Nästa tal? ";
    if (!(cin >> i))
        break;
    int r = f(i);
    cout << "Resultat: " << r << endl;
}
catch(overflow_error) {
    cout << 'Resultat utanför giltigt område');
}
catch(exception& e) {
    cout << typeid(e).name() << ": " << e.what() << endl;
}
```

Att fånga exceptionella händelser

Exempel:

```
int i;
try {
    cout << "Nästa tal? ";
    if (!(cin >> i))
        break;
    int r = f(i);
    cout << "Resultat: " << r << endl;
}
catch(overflow_error) {
    cout << 'Resultat utanför giltigt område');
}
catch(exception& e) {
    cout << typeid(e).name() << ": " << e.what() << endl;
}
```

Fördefinierad funktion i klassen exception

Specifikation av exceptionella händelser

I en funktion kan typerna för de händelser som kan genereras av funktionen specificeras i en *händelselista*

Exempel på händelselista:

```
int f(int) throw(typ1, typ2, typ3) {
    //...
    throw typ1("Fel av typ 1 har inträffat");
    throw typ2("Fel av typ 2 har inträffat");
    throw typ3("Fel av typ 3 har inträffat");
    //...
}
```

Ingen lista	==>	Alla typer av händelser kan genereras
Tom lista (`throw()`)	==>	Inga händelser kan genereras

Specifikation av exceptionella händelser

Händelselistan måste finnas både i deklarationen

```
// Dekl. i klassen Vektor
int& operator[] (int i) throw(out_of_range);
```

och i definitionen av funktionen

```
// Def.
int& Vektor::operator[] (int i) throw(out_of_range) {
    if (i<0 || i>=ant)
        throw out_of_range("Vektor::operator[]");
    return p[i];
}
```