Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Variabilne šablone

Nejc Maček

Mentorji:

- doc. dr. Matej Črepinšek
- doc. dr. Tomaž Kosar
- red. prof. dr. Marjan Mernik

Maribor, 7. 3. 2016

Kazalo vsebine

Uvod

Variabilne šablone (*variadic templates*) so funkcionalnost, ki jo lahko zasledimo v jezikih C++ in D. Je razširitev navadnih šablon s posebnim parametrom, ki omogoča prejemanje poljubnega števila argumentov. Jeziku C++ je bila ta funkcionalnost dodana v verziji C++11.

Posvetili se bomo variabilnim tabelam v jeziku C++.

Primerjava in podobnosti v drugih jezikih

Veliko programskih jezikov podpira definicije funkcij, ki lahko prejmejo poljubno število elementov. To število je znano le ob klicu funkcije, pri implementaciji pa je neznano. Pri deklaraciji funkcije se v večini primerov takšen način prejemanja argumentov označi s posebnim operatorjem, ki stoji poleg zadnjega parametra v predpisu. Sintaksa klica funkcije je povsem običajna, le da lahko takim funkcijam pošljemo poljubno število argumentov.

C#

```
static void PrintAll(
     params string[] args) {
     foreach (var arg in args)
          Console.WriteLine(arg);
}
```

Java

```
static void printAll(String... args) {
     for (String arg : args)
          System.out.println(arg);
}
```

JavaScript (ECMAScript 5)

```
function printAll() {
     for (var i = 0;
          i < arguments.length;
          i++)
          console.log(arguments[i]);
}
```

JavaScript (ECMAScript 6)

```
function printAll(...args) {
     for (let arg of args)
          console.log(arg);
}
```

Ruby

```
def printAll(*args)
     for arg in args
          print arg, "\n"
     end
end
```

Python

```
def printAll(*args):
     for arg in args:
          print(arg)
```

V primeru jezika JavaScript (ECMAScript) lahko zasledimo, da nova verzija poenostavi in izboljša omenjeno funkcionalnost. Poleg tega vsebuje še druge podobne funkcionalnosti, na primer razširjanje polj (*array*).

Opazimo lahko, da se »dodatni« argumenti shranijo kar v polje. Pri jezikih, ki podpirajo polja različnih tipov, lahko zato funkciji pošljemo argumente različnih tipov.

Seznam argumentov

Možnost podajanja poljubnega števila argumentov neki funkciji zasledimo tudi pri jeziku C++, (ne da bi uporabljali variabilne šablone). Imenuje se seznam argumentov (*argument list*). Sintaksa deklaracije funkcije uporablja tropičje, ki sledi vsaj enemu argumentu. V določenih različicah nekaterih prevajalnikov mora temu operatorju slediti tudi vejica. Do posameznih argumentov se dostopa z `va_list`, `va_start`, `va_arg`, in `va_end`, kot kaže primer.

```
void printAll(int count, ...) {
     va_list list;
     va_start(list, count);
     for (int i = 0; i < count; i++) {
          int arg = va_arg(list, int); // pričakujemo argumente tipa int
          cout << arg << endl;
     }
     va_end(list);
}
```

Vidno je, da ima takšen način vnosa podatkov kar nekaj slabosti.

- Pred tropičjem mora biti vsaj en parameter.
- Da lahko dostopamo do vseh argumentov, moramo poznati njihovo število.
- Zahtevati moramo pravilni tip argumenta, sicer lahko pride do nepravilnosti delovanja programa. Tip naslednjega argumenta podamo makru `va_arg` kot drugi argument.

Tak način podajanja argumentov je primeren le v redkih primerih – kjer imamo znano število argumentov in njihov tip. Ti vrednosti sta po navadi vneseni preko ostalih, »obveznih«, argumentov. Ker pa sta zaradi napake programerja lahko neskladni z dejanskimi argumenti2013, ta funkcionalnost ni »tipovsko« varna (*type-safe*). Primer uporabe seznama argumentov je funkcija `printf`.

V primeru da imamo opravka z argumenti istega tipa, je smiselno in priporočljivo tega uporabiti polje ali kak drug podatkovni tip.

Variabilne šablone

Variabilne šablone (*variadic templates*) so šablone, ki so idealna rešitev za funkcije, katerim želimo poslati poljubno število argumentov, saj nam omogočajo ravno to. Z njihovo uporabo se znebimo nevšečnosti, ki se pojavijo pri seznamu argumentov.

Paket parametrov

Paket parametrov (*parameter pack*) je lahko ali šablonski paket parametrov ali funkcijski paket parametrov. Zapisuje se s pomočjo tropičja (`...`), ki ni vezano na neberljive znake (*whitespace*) za ali pred njim.

Šablonski paket parametrov

Variabilne šablone imajo poseben parameter imenovan šablonski paket parametrov (*template parameter pack*). Temu lahko podamo poljubno število argumentov; lahko tudi nobenega. Zapišemo ga s tropičjem, ki je stoji za ključno besedo, ki definira vrsto tipa ali tip parametra. Izbirno lahko za tropičjem navedemo tudi ime šablonskega paketa parametrov.

```
template <typename ...>

template <class ... Types>

template <int ... Values>

template <template <class T> class ... Classes>
```

Variabilne šablone uporabimo kot navadne, le da jim lahko podamo poljubno mnogo argumentov.

```
template <typename ... Types>
class C1;

template <typename Base, typename ... Others>
class C2;

template <int ...>
class C3;

C1<>                    c1_1;    // Types je brez parametrov
C1<int>                 c1_2;    // Types = itn
C1<int, double>         c1_3;    // Types = int, double
C1<0>                   c1_4;    // napaka (0 ni tip)

C2<>                    c2_1;    // napaka (tip Base mora biti določen)
C2<int>                 c2_2;    // Base = int; Types je brez parametrov
C2<int, double>         c2_3;    // Base = int; Types = double
C2<int, double, float>  c2_4;    // Base = int; Types = double, float

C3<>                    c3_1;    // brez vrednosti
C3<int>                 c3_2;    // napaka (int ni vrednost)
C3<0>                   c3_3;    // vrednost:  0
C3<0, 1>                c3_4;    // vrednosti: 0, 1
```

Funkcijski paket parametrov

Funkcijski paket parametrov je parameter, ki ga uporabimo pri deklaraciji funkcije deklarirane s pomočjo variabilne šablone ali metode v razredu oz. strukturi deklarirani s pomočjo take šablone. Tudi tega zapišemo s tropičjem. To mora v predpisu funkcije stati za imenom deklariranega tipa šablonskega paketa parametrov in pred imenom funkcijskega paketa parametrov.

```
template <class ... Ts>
void fn1(Ts ... args);
```

Dolžina paketa parametrov

Dolžine paketa parametrov sicer pri deklaracije funkcije ne moremo določiti, je pa njegova dolžina znana ob klicu funkcije oz. ustvarjanju razreda. Zato lahko v sami deklaraciji funkcije pridobimo število argumentov, ki bo podano šablonskemu paketu parametrov in s tem tudi število argumentov podanih funkcijskemu paketu parametrov, ki je pravzaprav enako. V ta namen nam služi operator `sizeof...`, kateremu kot argument podamo funkcijski paket parametrov, ne da bi ga pri tem razširili. Vrnjena vrednost je tipa `unsigned int`.

```
template <typename ... Ts>
void fn2(Ts ... args) {
     cout << "Prejetih " << sizeof...(args) << " argumentov." << endl;
}
```

Več parametrov

Šablonski paket parametrov lahko le en izmed več parametrov šablone. V primarni deklaraciji šablone razreda mora biti podan zadnji, medtem ko ga lahko v šabloni funkcije nasledi kak drug parameter, katerega tip je nedvoumno določljiv iz parametrov funkcije.

```
template <class T, class ... Rs> // v redu (Rs je zadnji parameter šablone)
class C4;

template <class ... Ts, class R> // napaka (Ts ni zadnji parameter šablone)
class C5;

template <class T, class ... Rs> // v redu (Rs je zadnji parameter šablone)
void fn3();

template <class ... Ts, class R> // napaka (dvoumno)
void fn4();

template <class ... Ts, class R> // v redu (nedvoumno)
void fn5(Ts ... first, R last);
```

Razširjanje paketov

Paket parametrov uporabimo tako, da ga razširimo. Temu pravimo razširitev paketa (*pack expansion*) in sestoji iz vzorca (*pattern*) in tropičja, ki je zapisan za vzorcem. Vzorec je po navadi kar sam parameter. Razširimo lahko tako šablonski kot tudi funkcijski paket parametrov. Pri tem velja pravilo, da bo rezultat razširjanja smatran kot z vejico ločen zapis vzorca razširitve, ki mu bo namesto paketa parametra dana dejanska vrednost zaporednega izmed argumentov, (ki so bili podanih paketu parametrov).

```
// primer razširitve šablonskega paketa
template <class ... Ts>
class C6;

template <class ... Ts>
class C7 {
     void fn() {
          C6<Ts...> c6; // C6<Ts1, Ts2, Ts2, ..., Tsn> c6;
     }
};

// primer razširitve funkcijskega paketa
void fn6(Ts ... args) {
     fn1(args...); // fn1(arg1, arg2, arg3, ..., argn);
}
```

Razširjanje šablonskega paketa parametrov

Z razširjanjem šablonskega paketa v šabloni razreda lahko določimo, od katerih razredov bo naš novi razred dedoval. V takih primerih se po navadi tudi v konstruktorju pojavi funkcijski paket parametrov, ki je uporabljen za inicializacijo starševskih razredov. Lahko pa bi pri vsakem osnovnem razredu klicali kar privzeti konstruktor.

```
template <class ... Classes>
class C8 : Classes... {
public:
     C8() : Classes()... {}

     template <typename ... Ts>
     C8(Ts ... params) : Classes(params)... {}
};
```

Razširjanje funkcijskega paketa parametrov

Z razširjanjem funkcijskih paketov parametrov pridejo v večji poštev vzorci. Razširjanje takega paketa bo razširilo najobsežnejši možni vzorec levo od tropičja.

```
template <class ... Ts>
void fn7(Ts ... args) {                 // predpostavimo klic fn7(1, 2, 3);
     fn1(args...);                      // fn1(1, 2, 3);
     fn1(++args...);                    // fn1(++1, ++2, ++3);
     fn1(fn(args)...);                  // fn1(fn(1), fn(2), fn(3));
     fn1(fn(args...) + args...);        /* fn1(fn(1, 2, 3) + 1,
                                               fn(1, 2, 3) + 2,
                                               fn(1, 2, 3) + 3); */
     fn1((args + 1)... + args...);      /* fn1(1 + 1, 2 + 1, 3 + 1 + 1,
                                               1 + 1, 2 + 1, 3 + 1 + 2,
                                               1 + 1, 2 + 1, 3 + 1 + 3); */
}
```

Prevajanje

Vsi argumenti podani paketom parametrov se razširijo v času prevajanja in se v prevedenem programu izvajajo kot funkcije s točno določenimi argumenti.

Če pri prevajanju pride do napak pri neujemanju argumentov ali dolžine paketa parametrov, prevajalnik javi napako.

```
void fn8(int i1, int i2);

template <typename ... Ts>
void fn9(Ts ... args) {
     fn8(args...);
}

void fn10() {
     fn9(1, 2);         // v redu
     fn9(1, 2, 3);      // napaka – neujemanje števila argumentov funkcije fn8
     fn9("a", "b");     // napaka – neujemanje tipa argumentov funkcije fn8
}
```

Uporaba

Zaradi abstraktnosti in nedoločljivosti paketa parametrov je uporaba variabilnih šablon precej redka.

Argumente funkcijskega paketa parametrov lahko shranimo. To najlažje naredimo s pomočjo že definiranega razreda `tuple`. Funkcija `get` omogoča, da lahko pridobimo vrednost na določenem indeksu. Ta način uporabe ni najbolj pogost, saj če imamo poljubno število argumentov, nas verjetno ne zanima točno določeni izmed njih. Lahko pa na tak način preletimo vse vrednosti v paketu.

Za slednji pristop pa je bolj pogosta in priporočljiva uporaba rekurzije. Argumente nekega funkcijskega parametra lahko namreč razširimo in podamo funkciji, ki bo obdelala prvega izmed njih. To je možno s preprosto deklaracijo šablone, pri kateri je prvi parameter enolično določen, drugi pa je paket parametrov. Ta funkcija lahko s preostalimi argumenti (torej vsemi razen prvega) spet kliče sama sebe in na tak način izvaja rekurzijo, pri čimer bo v vsaki stopnji te rekurzije klicana z enim argumentom manj, vse do zadnjega preostalega argumenta.

```
template <typename T>
void print(T t) {
     cout << t << endl;
}

template <typename T, typename ... Rs>
void print(T t, Rs ... rest) {
     cout << t << endl;
     print(rest...);
}
```

Pogost način uporabe je lahko tudi ta, da je funkcija s funkcijskim paketom parametrov le »posrednik« med klicem te funkcije in klicem neke druge funkcije. Na tak način lahko ustvarimo tovarno (*factory*). Tak način uporabe prikazuje spodnji zgled, pri katerem še posebej pazimo, da vračamo kazalce do ustvarjenih objektov in ne objektov samih, za kar uporabimo strukturo `remove_pointer`.

```
template <class T, typename ID, typename ... Params>
class Factory {
private:
     using Tp  = typename remove_pointer<T>::type*;
     using Map = unordered_map<ID, Tp>;
     Map map;
     Tp createNew(ID id, Params ... args) {
          Tp t = new T(args...);
          map.insert({ id, t });
          return t;
     }
public:
     bool contains(ID id) {
          auto mi = map.find(id);
          return mi != map.end();
     }
     Tp get(ID id) {
          auto mi = map.find(id);
          if (mi == map.end()) return nullptr;
          return map.at(id);
     }
```

```
    Tp create(ID id, Params ... args) {
        if (contains(id)) {
            return get(id);
        } else {
            return createNew(id, args...);
        }
    }
    ~Factory() {
        for (auto const i : map) {
            delete i.second;
        }
    }
};
```

Zaključek

Variabilne šablone zaradi svoje abstraktnosti niso pogosto uporabljene, vendar so v določenih primerih zelo uporabne, saj lahko avtomatizirajo pristope, ki niso odvisni od tipa in števila argumentov. Vsekakor je njihova uporaba priporočljiva, še posebej kadar se z njimi lahko izognemo slabim pristopom programiranja, kot je na primer seznam argumentov.

Viri

Allain, A. (27. 2 2016). *Functions with Variable Argument Lists in C and C++ using va_list.* Pridobljeno iz C programming.com: http://www.cprogramming.com/tutorial/lesson17.html

Building factories in C++ with boost::factory. (4. 8 2015). Pridobljeno 6. 3 2016 iz https://www.youtube.com/watch?v=m5snRgBLC0U

C++ International Standard. (5. 3 2016). Pridobljeno iz Open standards: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3797.pdf

D Programming Language. (5. 3 2016). Pridobljeno iz D Programming Language: http://dlang.org/

Ellipses and Variadic Templates. (4. 3 2016). Pridobljeno iz Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/dn439779.aspx

Functions with Variable Argument Lists. (4. 3 2016). Pridobljeno iz Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/fxhdxye9.aspx

Korpela, H. (5. 3 2016). *C++11 - New features - Variadic templates.* Pridobljeno iz cplusplus.com - The C++ Resources Network: http://www.cplusplus.com/articles/EhvU7k9E/

Parameter pack. (4. 3 2016). Pridobljeno iz cppreference.com: http://en.cppreference.com/w/cpp/language/parameter_pack

sizeof... operator. (6. 3 2016). Pridobljeno iz cppreference.com: http://en.cppreference.com/w/cpp/language/sizeof...

Variable number of arguments in C++? (5. 3 2016). Pridobljeno iz Stack Overflow: http://stackoverflow.com/questions/1657883/variable-number-of-arguments-in-c