Funcții și Clase Template

Funcții şi clase template

- Un **template** (**şablon**) defineşte o familie de funcții, respectiv de clase generice (*parametrizate*) utilizând tipurile ca parametri.
- Reprezintă o altă *tehnică de reutilizarea a codului*.
- Utilizăm mecanismul **templates** atunci când avem nevoie de funcții / clase ce aplică *aceleaşi operații / instrucțiuni* (acelaşi algoritm) asupra mai multor tipuri de date.

Funcții şi clase template

- Template-urile pot fi considerate o continuare a noțiunii de macro.
- Utilizarea template-urilor generează cod în plus în etapa de compilare.
- Un template funcționează ca şi un macro, prin substituție.
  - substituția pentru o clasă nu se realizează în locul în care apare prima data o instanță a acesteia: se generează codul pentru o clasă substituită, şi toate instanțele ce folosesc acelaşi tip de date vor folosi acelaşi template substituit.

Funcții şi clase template

- Într-un template definim un model al unei funcții sau al unei clase în care tipurile de date cu care lucrăm pot fi stabilite din faza de precompilare.
- Deoarece substituția se face în faza de precompilare, template-urile vor fi amplasate în fişiere header (.h).

Problema 1

- Să se realizeze o funcție ce determină valoarea maximă dintr-un şir de elemente unde
  - elementele sunt numere întregi;
  - elementele sunt numere reale;
  - . . .

Soluție

Funcții identice

diferă tipul unor parametri

```
int max(int n, int t[]) {
  int m = t[0];

  for(int i = 1; i < n; i++) {
    if (t[i] > m) {
      m = t[i];
    }
  }
  return m;
}
```

```
float max(int n, float t[]) {
  float m = t[0];

  for(int i = 1; i < n; i++) {
    if (t[i] > m) {
      m = t[i];
    }
  }
  return m;
}
```

```
int main() {
  int a[3] = {2, 1, 9};
  float b[4] = {3.0, 1.2, 4.8, 1.9};

  cout << max(3,a) << endl;
  cout << max(4,b);

  return 0;
}
```

Funcții şi clase template

- mecanismul **templates** se află la baza *programării generice*.
- *programarea generică* ne permite să scriem clase și funcții ce sunt polimorfe pe tipuri ce nu au legătură între ele la momentul compilării.
  - o singură clasă sau funcție poate fi utilizată pentru a manipula obiecte dintr-o varietate de tipuri.

Exemplul 1

```
template <class T>
T max(int n, T t[]) {
  T m = t[0];
  for (int i = 1; i < n; i++) {
    if (t[i] > m) {
      m = t[i];
    }
  }
  return m;
}

int main() {
  int a[3] = {2, 1, 9};
  float b[4] = {3.0, 1.2, 4.8, 1.9};

  cout << max<int>(3,a) << endl;
  cout << max<float>(4,b) << endl;
  cout << max(3,a) << endl; //tipul este dedus din tipul elemen. lui a
  return 0;
}
```

Funcții template

- O **funcție template** (*şablon, generică)* este un tipar utilizat de compilator pentru a construi automat diverse funcții.
- Se utilizează pentru implementarea unor funcții ce diferă doar prin tipul parametrilor.
- Sintaxă:

```
template <par1, par2,..., parn>
antet_functie_parametrizata;
```

unde:

`par1, par2,..., parn` sunt parametrii funcției template, de regulă *constante* sau *tipuri de date* specificate prin cuvântul cheie class sau typename.

Parametrii din lista parametrilor şablonului (template) trebuie să apară toți în lista de parametri formali ai funcției template.

Apelul funcțiilor template

- Sintaxa apelului unei funcții template:

```
nume_functie_parametrizata(expr1, expr2,...,exprn)
```

unde **expr1**, **expr2**, ..., **exprn** sunt expresii din care se deduc tipurile concrete

sau

```
nume_functie_parametrizata<tipC1,tipC2,…, tipCn>(lista_parametri)
```

unde **tipC1**, **tipC2**, ..., **tipCn** sunt tipuri concrete sau constante utilizate pentru a genera versiunea corespunzătoare de funcție.

Generarea funcțiilor template

- **Generarea de cod** pentru *entitatea template* are loc *la compilare*.
- La apelul funcției parametrizate, tipul argumentelor determină ce versiune a şablonului este folosită.

Problema 2

- Să se elaboreze o clasă ce implementează noțiunea de **Vector** cu elemente numere reale.

Soluție

```
class Vector {
    float elem[MAX];
    int n;
  public:
    Vector(int n);
    int getNrElemente() const;
    float& operator[] (int i);
    void print();
};

Vector::Vector(int n) {
  this->n = n;
  if (n > MAX) {
    throw out_of_range("Depasire dimensiune
                        maxima");
  }
}
int Vector::getNrElemente() const {
  return n;
}
float& Vector::operator[] (int i) {
  if ((0 <= i) && (i < n)) {
    return elem[i];
  } else {
      throw out_of_range("Depasire limite");
  }
}

void Vector::print() {
  int i;
  for (i = 0; i < n; i++) {
    cout << elem[i] << " ";
  }
  cout << endl;
}

int main() {
  Vector v(5);

  for(int i = 0; i < v.getNrElemente(); i++) {
    v[i] = i / 2.0;
  }
  v.print();

  return 0;
}
```

Ce se întâmplă dacă dorim un tablou cu numere întregi sau cărțile dintr-o bibiotecă?

Clase Template

- O **clasă template** (*generică*) este un model (un *şablon*) utilizat pentru generarea unor clase concrete, clase ce diferă prin tipul anumitor date membre.
- Sintaxă:

```
template <par1, par2,..., parn> declarare_clasă;
```

`par1, par2,..., parn` sunt parametrii clasei template, de regulă *constante* sau *tipuri de date* specificate prin cuvântul cheie class sau typename.

Instanțierea claselor template

- Sintaxa instanțierii unei clase template:

```
nume_clasa <tipC1,tipC2,…, tipCn>
nume_obiect(lista_param_constructor);
```

unde **tipC1**, **tipC2,...**, **tipCn** sunt tipuri concrete sau constante utilizate pentru a genera versiunea corespunzătoare.

Exemplul 2

```
template <class T>
class Vector {
  private:
    T elem[MAX];
    int n;

  public:
    Vector(int n);
    int getNrElemente() const;
    T& operator[] (int i);
    void print();
};

template <class T>
Vector<T>:: Vector(int n) {
  this->n = n;
  if (n > MAX) {
    throw out_of_range("Depasire dimensiune
                        maxima");
  }
 }
```

```
template <class T>
int Vector<T>::getNrElemente() const {
  return n;
}

template <class T>
T& Vector<T>:: operator[] (int i) {
  if ((0 <= i) && (i < n)) {
    return elem[i];
  } else {
      throw out_of_range("Depasire limite");
  }
}

template <class T>
void Vector<T>::print() {
  int i;
  for (i = 0; i < n; i++) {
    cout << elem[i] << " ";
  }
  cout << endl;
}
```

Exemplu 2 (cont.)

```
int main() {
  Vector<int> vi(5);
  for(int i = 0; i < vi.getNrElemente(); i++) {
    vi[i] = i * i;
  }
  vi.print();

  Vector<float> vf(5);
  for(int i = 0; i < vf.getNrElemente(); i++) {
    vf[i] = i/2.0;
  }
  vf.print();

  return 0;
}
```

Exemplu 2 (cont.)

```
class Carte {
  private:
    char titlu[30];
    char autor[30];
  public:
    Carte(const char *titlu = "",
          const char *autor = "") {
      strcpy(this->titlu, titlu);
      strcpy(this->autor, autor);
    }

    friend ostream& operator<< (ostream& out,
                                const Carte& c);
};
```

```
ostream& operator<<(ostream& out, const Carte&c) {
  out << "Titlu:" << c.titlu << endl;
  out << "Autor:" << c.autor << endl;

  return out;
}


int main() {
  Vector<Carte> v(3);

  v[0] = Carte("Thinking in C++", "B. Eckel");
  v[1] = Carte("Limbajul C++", "I. Muslea");
  v[2] = Carte("The C++ standard library",
               "N. M. Josuttis");
  v.print();

  return 0;
}
```

Parametru template

- Un parametru al unui template poate fi:
  - orice tip de dată predefinit (`int`, `bool`, `double`, `char` etc.);
  - orice tip de dată definit de către utilizator;
  - o expresie constantă;
  - tipul `void`;
  - un tip de pointer;
  - un tip referință;
  - o funcție;
  - alt template.

Funcții şi clase template

- compilatorul generează o clasă sau o funcție specifică pe baza tipurilor prezentate ca argumente.
- un *şablon de clasă* (*class template*) = *un generator de tipuri*.
- procesul de generare a tipurilor de date dintr-un şablon de clasă se numeste *specializare* (*specialization*) sau *instanțierea șablonului* (*template instantiation*).

Funcții şi clase template

- *parametrii unui şablon* (*template parameters*) - se referă la identificatorii ce sunt enumerați după cuvântul cheie `template` într-o declarație a unui şablon.
- *argumentele unui şablon* (*template arguments*) - se referă la înlocuitorii parametrilor unui şablon în timpul operației de specializare.
- *instanțiere* (*instantiation*) - procesul în care compilatorul generează o clasă sau o funcție obişnuită prin înlocuirea fiecăruia dintre parametrii şablonului cu un tip concret.
- *instanțiere implicită* (*implicit instantiation*) - atunci când compilatorul decide când să genereze cod pentru instanțele template-urilor noastre.

Funcții şi clase template

- *instanțiere explicită* (*explicit instantiation*) - atunci când programatorul stabileşte când compilatorul ar trebui să genereze codul pentru o anumită specializare.
- *specializare* (*specialization*) - clasa sau funcția rezultată în urma procesului de instanțiere a unui şablon este o specializare.
- *specializare parțială* (*partial specialization*) - se referă la situația în care se specifică o specializare a şablonului pentru o submulțime dintre toate cazurile posibile. De exemplu, în cazul în care şablonul acceptă mai mulți parametri, îl putem specializa parțial prin definirea unui caz în care specificăm un tip concret doar pentru unul dintre parametri.

Concluzii

- utilizarea şabloanelor poate conduce la timpi de compilare mai lungi şi la un cod executabil mai mare.
- compilatoarele oferă adesea mesaje de eroare şi de diagnosticare greu de înțeles şi de depanat.
- modul în care au fost proiectate colecțiile STL conduce adesea la crearea unei mulțimi de copii de obiecte.
- pe de altă parte, avem containere şi algoritmi generici siguri şi eficienți.
- motivele principale ale utilizării şabloanelor se referă la performanță și întreținerea mai uşoară a unei aplicații.

Temă

- Implementați clasa `Multime` care să poată reține elemente de orice tip. Să se supraîncarce operatorii ‘`+`’, ‘`*`’ pentru a realiza reuniunea respectiv intersecția a două mulțimi.
- Implementați o clasa generică (template) care să implementeze un **arbore binar**.

Referințe

- http://users.cis.fiu.edu/~weiss/Deltoid/vcstl/templates
- https://www.cs.bham.ac.uk/~hxt/2016/c-plus-plus/intro-to-templates.pdf
- https://mariusbancila.ro/blog/2021/03/15/typename-or-class/
- https://conradsanderson.id.au/misc/sanderson_templates_lecture_uqcomp7305.pdf
- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-s096-introduction-to-c-and-c-january-iap-2013/lectures-and-assignments/c-introduction-classes-and-templates/