

LUCRAREA DE LABORATOR NR 5

Tema: Funcții și clase generice

Scopul lucrării:

- Studierea necesității șabloanelor (funcțiilor și claselor generice);
- Studierea regulilor de definire și utilizare a șabloanelor;
- Studierea specializării șabloanelor;
- Studierea potențialelor probleme rezolvate cu ajutorul șabloanelor;

Noțiuni generale

Motivarea

Șabloanele reprezintă cea mai puternică construcția a limbajului C++, dar în același timp, unul din cele mai puțin studiate și rar utilizat. Cauza este ascunsă în faptul că el este foarte complicat și are o sintaxă neobișnuită.

Așa deci, șabloanele reprezintă prin sine un mecanism ce permite să scrii un algoritm, care nu este legat de un tip anumit. Cel mai des, șabloanele sunt utilizate pentru crearea container și algoritmi abstracți. Containerele sunt obiecte, care conțin alte obiecte sau date, cel mai des o cantitate nedeterminată, așa cum sunt masivele, stivele, liste asociative, etc. Prin algoritm abstract este necesar de înțeles studierea bună a metodelor de prelucrare a datelor, ca sortarea, căutarea, etc., înscrise fără prezentarea tipului datelor.

Șabloanele sunt clase și funcții. Șabloanele au venit să schimbe macrourile, așa cum ultimele foarte des duc la găsirea complicată a erorilor, deoarece compilatorul nu verifică, dar nici nu are posibilitatea să le verifice de erori sintactice.

Programatorul, scriind șabloanele, creează aprovizionarea, care, ca urmare, se utilizează deja cu tipurile de date specificate. Adică, la baza șabloanelor compilatorul creează funcții normale. Dacă șabloanele sunt utilizate cu câteva tipuri de date diferite, compilatorul creează un codul necesar pentru fiecare tip în parte. Cu alte cuvinte, șabloanele nu micșorează compilarea modulului, mai degrabă chiar invers, dar simțitor micșorează codul de ieșire, care duce la, micșorarea cantității de erori, micșorează introducerea modificărilor în cod și micșorează prezentarea programelor în general, așa cum se micșorează calitatea tipurilor și funcțiilor date.

Definirea

Șabloanele (funcții și clase generice) sunt definite cu ajutorul cuvântului rezervat *template*:

Pentru funcția generică:

```
template <typename T>
T& searchmax(T* ptr, int size)
{
    ...
}
```

Pentru clasa generică:

```
template <typename T>
class Stack{
    T mas[10];
public:
    ...
};
```

Din exemplu se vede specificul definirii șablonului, pentru crearea lui este necesar de prezentat cuvântul *template*, în paranteze unghiulare cuvântul *class*¹(sau *typename*) și un tip generic (parametrizat), care se va utiliza în definirea șablonului. Istoric așa sa întâmplat, că cel

¹ În corespundere cu noile standarde, la fel poate fi utilizat “*typename*” în locul “*class*”.

mai des, se utilizează identificatorul *T*, de la cuvântul *type*. Pentru clasă, la definirea funcțiilor în afara clasei, înainte de fiecare funcție este necesar de scris, din nou, cuvântul *template*.

Utilizarea

Funcțiile se utilizează practic așa ca și funcțiile obișnuite.

```
void main() {
    int    masi[10];
    float  masf[20];
    cout<<searchmax(masi,10);
    cout<<searchmax(masf,20);
}
```

Se vede că, sintaxa, apelării coincide cu cele obișnuite. Cunoscând tipurile parametrilor funcțiilor, compilatorul generează funcții obișnuite. După care are posibilitatea să supraîncarce funcția. În cazul claselor lucrurile sunt puțin mai complicate:

```
void main() {
    Stack<int>    sti;
    Stack<float> stf;
}
```

Este necesar de prezentat tipul datelor, așa cum, în acest caz, la această etapă de translare, compilatorul nu este în stare să determine, pentru care tip de date este necesar de generat codul.

Specializarea

Câte odată nu ne satisface lucrul șabloanelor pentru determinarea tipurilor datelor. Ca de exemplu:

```
template <class T>
T& max(T& a, T& b) {
    if(a>b) return a;
    return b;
}
```

Acest exemplu lucrează excelent pentru tipurile încorporate, așa ca *int*, *float* și altele. Dar pentru șiruri – nu. Motivul constă în aceea, că în acest caz se vor compara pointerii la șiruri, dar nu conținutul șirului. Pentru alte tipuri de date, posibil nu este definit operatorul *>*. Rezolvări pot fi câteva: se poate de interzis utilizarea pointerilor la șiruri și să utilizezi tipul *String*, pentru care este definit operatorul *>*, atunci în acest caz se complică procesul de dezvoltare și regulile de utilizare. Mai mult ca atât, interzicerea poate fi numai informativă, adică dacă utilizatorul știe, că nu trebuie de utilizat pointeri. Însăși limbajul nu are posibilitatea să interzică utilizarea specificării unui tip de date special. Altă rezolvare constă în utilizarea specializării, care reprezintă înscrierea încă a unei funcții pentru un tip determinat. În cazul funcțiilor această de obicei nu este o funcție șablon cu același nume și cu parametri predefiniți. Această funcție poate avea un avantaj mai mare decât șabloanele. În cazul claselor se poate desigur de definit o clasă neșablon cu același nume, dar aceasta nu este interesant, așa cum deosebiriile pot fi minimale. În acest caz poate fi utilizat specializarea metodei clasei. Specializat poate fi numai metoda definit în afara clasei. Ca de exemplu:

```
template <class T>
class Stack{
public:
    void push(T& t);
    void sort();
friend ostream& operator<<(ostream& os, Stack<T>& s);
};
```

```

template <class T>
void Stack<T>::sort() {
    ...
    // aici se înscrie un algoritm abstract
}
void Stack<char*>::sort() {
    ... // dar aici unul specializat
}
template <class T>
ostream& operator<<(ostream& os, Stack<T>& s) {
    return os; // afișarea conținutului stivei
}
void main() {
    Stack<int> si;
    si.push(5);
    Stack<char*> sc;
    sc.push("Hello");
    si.sort(); // Apelarea funcției abstracte
    sc.sort(); //Apelarea funcției specializate
    cout<<si<<sc;
// Apelarea operatorului de supraîncărcare a fluxului de ieșire
}

```

Șabloanele clasei pot fi moștenite, așa ca și cele obișnuite, cu aceasta, și acel de bază, așa și a cel derivat pot fi clase obișnuite.

```

class One{
};
template <class T>
class Two: public One{
};
template <class T>
class Three: public Two<T>{
};
class Four: public Three<int>{
};
template <class T>
class Five: public T{
};

```

Un interes deosebit reprezintă prin sine ultimul tip de moștenire, așa cum are loc moștenirea de la parametrii șablonului. În acest caz *T* desigur nu trebuie să fie clasă ori structură.

Cum se vede din cele relatate mai sus, șabloanele reprezintă prin sine un mecanism interesant și puternic. Sunt greuțâți și complicații în studierea lor, dar ele se răscumpără, așa cum permite crearea unui cod frumos și compact. Mai mult ca atât, șabloanele nu sunt realizate în alte limbaje de programare moderne utilizate pe larg, dar la fel permit realizarea unor noi rezolvări tehnice, așa cum pointeri deștepți, susținerea tranzacției, dirijarea memoriei, etc.

Întrebări de control:

1. Ce reprezintă prin sine șabloanele?
2. Care sunt avantajele utilizării șabloanelor?
3. Pentru ce se utilizează specializarea?
4. Ce poate fi un șablon?
5. Care sunt problemele utilizării șabloanelor?
6. Utilizarea șabloanelor duce la economia memoriei?
7. Dar a timpului de executare a compilării codului?
8. Poate o clasă șablon să moștenească una obișnuită și invers?

Sarcina

Varianta 1

- a) Creați o funcție generică (șablon), care schimbă ordinea elementelor în felul următor: prima parte a listei se amestecă la urmă, dar a doua la început. De exemplu: 1 2 3 4 5 6 - 4 5 6 1 2 3. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite. Dacă numărul de elemente este impar, atunci elementul mijlociu nu trebuie de prelucrat.
- b) Creați clasa generică (parametrizată) *Stack*. Clasa trebuie să conțină constructorii, destructorii, și deasemenea funcțiile *push*, *pop*, *empty*, *full* și operatorii de intrare/ieșire. Pentru alocarea memoriei să se utilizeze operatorul *new*.

Varianta 2

- a) Creați o funcție generică (șablon), care schimbă ordinea elementelor după perechi. De exemplu: 1 2 3 4 5 6 - 2 1 4 3 6 5. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite. Dacă numărul de elemente este impar, atunci ultimul element nu trebuie de prelucrat.
- b) Creați clasa generică (parametrizată) *Vector*. Clasa trebuie să conțină constructorii, destructorii, și deasemenea funcțiile *getLength*, operatorii *[]*, *+*, *-* și operatorii de intrare/ieșire. Pentru alocarea memoriei să se utilizeze operatorul *new*.

Varianta 3

- a) Creați o funcție generică (șablon), care calculează cantitatea de repetări a unui parametru în listă. De exemplu: lista - 0 2 3 4 3 6, parametrul - 3, rezultatul - 2. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.
- b) Creați clasele generice (parametrizate) *List* și *ListItem*. Clasa trebuie să conțină constructorii, destructorii și funcțiile *add*, *in*, *remove*, *getLength*, și operatorii de intrare/ieșire.

Varianta 4

- a) Creați o funcție generică (șablon), care caută o cheie dată. Funcția întoarce poziția primului element întâlnit. De exemplu: lista - 0 2 3 4 3 6, parametrul - 2, rezultatul - 1. În cazul când lipsește elementul necesar să se întoarcă codul erorii. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.
- b) Creați clasa generică (parametrizată) *Queue* - coadă. Clasa trebuie să conțină constructorii, destructorii și funcțiile *add*, *in*, *get*, *getLength*, operatorii de intrare/ieșire.

Varianta 5

- a) Creați o funcție generică (șablon), care caută al doilea element după dimensiune în elementele listei. De exemplu: lista - 0 2 3 4 3 6, rezultatul - 4. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.
- b) Creați clasa generică (parametrizată) *Set* - mulțimea. Clasa trebuie să conțină constructorii, destructorii și funcțiile *add*, *in*, *remove*, operatorii *“+”*, *“*”*, *“-”* și operatorii de intrare/ieșire.

Varianta 6

- a) Creați o funcție generică (șablon), de sortare în creștere după metoda bulelor. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.
- b) Creați clasa generică (parametrizată) *Map* – listă asociativă, care conține cheia câmpului și valoarea. Unei chei îi corespunde o valoare. Clasa trebuie să conțină constructorii, destructorii și funcțiile *add*, *removeByKey*, *getLength*, *getByKey*, *getByValue* și operatorii de intrare/ieșire.

Varianta 7

- a) Creați o funcție generică (șablon), care schimbă după perechi elementele tablouri unidimensionalelor în felul următor: primul element va avea valoarea sumei perechii, dar al doilea diferenței perechii.
De exemplu: lista- 0 2 3 4 3 6, rezultatul 2 -2 7 -1 9 -3.
- b) Creați clasa generică (parametrizată) *Matrix* – matrice. Clasa trebuie să conțină constructorii, destructorii și funcțiile *getRows*, *getCols*, *+*, *-*, *** și operatorii de intrare/ieșire.

Varianta 8

- a) Creați o funcție generică (șablon), care modifică ordinea elementelor invers. De exemplu: 1 2 3 4 5 6 - 6 5 4 3 2 1. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.
- b) Creați clasa generică (parametrizată) *Set* - mulțimea. Clasa trebuie să conțină constructorii, destructorii și funcțiile *add*, *in*, *remove*, operatorii *“+”*, *“*”*, *“-”* și operatorii de intrare/ieșire.

Varianta 9

- a) Creați o funcție generică (șablon), care calculează cantitatea de elemente, valoarea cărora este mai mare decât a parametrului dat. De exemplu: lista - 0 2 3 4 3 6, parametrul - 5, rezultatul - 1. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.
- b) Creați clasa generică (parametrizată) *Stack*. Clasa trebuie să conțină constructorii, destructorii și funcțiile *push*, *pop*, *empty* și operatorii de intrare/ieșire. Pentru alocarea memoriei să se utilizeze operatorul *new*, pe *n* elemente.

Varianta 10

- a) Creați o funcție generică (șablon), de sortare a elementelor unui masiv în descreștere prin metoda de introducere. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.
- b) Creați clasa generică (parametrizată) *Matrix* – matrice. Clasa trebuie să conțină constructorii, destructorii și funcțiile *getRows*, *getCols*, operatorii *+*, *-*, *** și operatorii de intrare/ieșire.

Varianta 11

- a) Creați o funcție generică (șablon), care calculează cantitatea de elemente, valoarea cărora este mai mică decât a unui element dat. De exemplu: lista - 0 2 6 4 3 3, parametrul - 3, rezultatul - 2. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.

b) Creați clasa generică (parametrizată) *Vector*. Clasa trebuie să conțină constructorii, destructorii, și deasemenea funcțiile *getLength*, operatorii *[]*, *+*, *-*, *** și operatorii de intrare/ieșire. Pentru alocarea memoriei să se utilizeze operatorul *new*.

Varianta 12

a) Creați o funcție generică (șablon), de căutare a elementului al doilea minimal după mărime din elementele listei. De exemplu: lista - 0 2 3 4 3 6, rezultatul - 2. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.

b) Creați clasa generică (parametrizată) *Matrix* – matrice. Clasa trebuie să conțină constructorii, destructorii și funcțiile *getRows*, *getCols*, *+=*, *-=*, **=* și de intrare/ieșire.

Varianta 13

a) Creați o funcție generică (șablon), care schimbă ordinea elementelor după perechi. De exemplu: 1 2 3 4 5 6 - 2 1 4 3 6 5. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite. Dacă numărul de elemente este impar, atunci ultimul element nu trebuie de prelucrat.

b) Creați clasa generică (parametrizată) *Vector*. Clasa trebuie să conțină constructorii, destructorii, și deasemenea funcțiile *getLength*, operatorii *[]*, *+*, *-* și operatorii de intrare/ieșire. Pentru alocarea memoriei să se utilizeze operatorul *new*.

Varianta 14

a) Creați o funcție generică (șablon), care caută al doilea element după dimensiune în elementele listei. De exemplu: lista - 0 2 3 4 3 6, rezultatul - 4. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.

b) Creați clasa generică (parametrizată) *Set* - mulțimea. Clasa trebuie să conțină constructorii, destructorii și funcțiile *add*, *in*, *remove*, operatorii *“+”*, *“*”*, *“-”* și operatorii de intrare/ieșire.

Varianta 15

a) Creați o funcție generică (șablon), care schimbă după perechi elementele tablouri unidimensionalelor în felul următor: primul element va avea valoarea sumei perechii, dar al doilea diferenței perechii.
De exemplu: lista- 0 2 3 4 3 6, rezultatul 2 -2 7 -1 9 -3.

b) Creați clasa generică (parametrizată) *Matrix* – matrice. Clasa trebuie să conțină constructorii, destructorii și funcțiile *getRows*, *getCols*, *+*, *-*, *** și operatorii de intrare/ieșire.

Varianta 16

a) Creați o funcție generică (șablon), care calculează cantitatea de elemente, valoarea cărora este mai mare decât a parametrului dat. De exemplu: lista - 0 2 3 4 3 6, parametrul - 5, rezultatul - 1. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.

b) Creați clasa generică (parametrizată) *Stack*. Clasa trebuie să conțină constructorii, destructorii și funcțiile *push*, *pop*, *empty* și operatorii de intrare/ieșire. Pentru alocarea memoriei să se utilizeze operatorul *new*, pe *n* elemente.

Varianta 17

- a) Creați o funcție generică (șablon), care calculează cantitatea de elemente, valoarea cărora este mai mică decât a unui element dat. De exemplu: lista - 0 2 6 4 3 3, parametrul - 3, rezultatul - 2. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.
- b) Creați clasa generică (parametrizată) *Vector*. Clasa trebuie să conțină constructorii, destructorii, și deasemenea funcțiile *getLength*, operatorii *[]*, *+=*, *-=*, **=* și operatorii de intrare/ieșire. Pentru alocarea memoriei să se utilizeze operatorul *new*.

Varianta 18

- a) Creați o funcție generică (șablon), de căutare a elementului al doilea minimal după mărime din elementele listei. De exemplu: lista - 0 2 3 4 3 6, rezultatul - 2. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.
- b) Creați clasa generică (parametrizată) *Matrix* – matrice. Clasa trebuie să conțină constructorii, destructorii și funcțiile *getRows*, *getCols*, *+=*, *-=*, **=* și operatorii de intrare/ieșire.

Varianta 19

- a) Creați o funcție generică (șablon), care calculează cantitatea de elemente, valoarea cărora este mai mică decât a unui element dat. De exemplu: lista - 0 2 6 4 3 3, parametrul - 3, rezultatul - 2. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.
- b) Creați clasa generică (parametrizată) *Vector*. Clasa trebuie să conțină constructorii, destructorii, și deasemenea funcțiile *getLength*, operatorii *[]*, *+*, *-*, *** și operatorii de intrare/ieșire. Pentru alocarea memoriei să se utilizeze operatorul *new*.

Varianta 20

- a) Creați o funcție generică (șablon), care schimbă ordinea elementelor în felul următor: prima parte a listei se amestecă la urmă, dar a doua la început. De exemplu: 1 2 3 4 5 6 - 4 5 6 1 2 3. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite. Dacă numărul de elemente este impar, atunci elementul mijlociu nu trebuie de prelucrat.
- b) Creați clasa generică (parametrizată) *Stack*. Clasa trebuie să conțină constructorii, destructorii, și deasemenea funcțiile *push*, *pop*, *empty*, *full* și operatorii de intrare/ieșire. Pentru alocarea memoriei să se utilizeze operatorul *new*.

Varianta 21

- a) Creați o funcție generică (șablon), care caută o cheie dată. Funcția întoarce poziția primului element întâlnit. De exemplu: lista - 0 2 3 4 3 6, parametrul - 2, rezultatul - 1. În cazul când lipsește elementul necesar să se întoarcă codul erorii. Funcția trebuie să lucreze cu tablouri unidimensionale de lungimi diferite.
- b) Creați clasa generică (parametrizată) *Queue* - coadă. Clasa trebuie să conțină constructorii, destructorii și funcțiile *add*, *in*, *get*, *getLength*, operatorii *[]* și operatorii de intrare/ieșire.

Varianta 22

- a) Creați o funcție generică (șablon), care schimbă după perechi elementele tablouri unidimensionalelor în felul următor: primul element va avea valoarea sumei perechii, dar al doilea diferenței perechii. De exemplu: lista- 0 2 3 4 3 6, rezultatul 2 -2 7 -1 9 -3.
- b) Creați clasa generică (parametrizată) *Matrix* – matrice. Clasa trebuie să conțină constructorii, destructorii și funcțiile *getRows*, *getCols*, +, -, * și operatorii de intrare/ieșire.