ШАБЛОНИ НА ТИПОВЕ

1. Цел на упражнението

Усвояване на технологията за използване на шаблони на функции и шаблони на класове.

2. Обща характеристика на шаблоните на типове

Повечето съвременни компилатори на C++ поддържат "супер-макро-механизъм", който позволява на програмистите да дефинират обобщени (с общо приложение) функции или класове, на базата на хипотетичен аргумент. Кодът на обобщените функции или класове се конкретизира когато техните дефиниции се използват с реални обекти. Обобщените дефиниции на функциите или на класовете се наричат ***шаблони*** (**templates**). Това е уникална възможност на C++, при която тип, а не обект от даден тип е аргумент.

Общата декларация на параметризиран тип започва със следната конструкция:

```
template <class имe_тип>
декларация_на_програмна_единица
```

където

- **template** е ключова дума, показваща, че в декларацията типът не е фиксиран;
- ***име_тип*** – произволен идентификатор, който в декларацията на програмната единица се използва като име на тип (типът може да бъде вграден или дефиниран от програмиста);
- ***декларация_на_програмна_единица*** е стандартна декларация на клас или дефиниция на функция.

След като бъдат дефинирани, шаблоните се използват, като се определя име, следвано от аргумент:

```
име_на_шаблон <аргументи>
```

3. Шаблони на функции

Дефинирането шаблон на функция е много подобно на дефинирането на конкретна функция. Разликата е, че аргументите на функцията са именовани по символичен начин. Използва се следният синтаксис:

```
template <class тип>
дефиниция_на_функция
```

където

- **<class тип>** е списък на параметризирани типове
- **тип** е идентификатор за формално име на тип, използвано в дефиницията на функцията;
- **дефиниция_на_функция** е стандартна дефиниция на функция със заглавна част и тяло.

4. Шаблони на класове

Параметризиран клас се декларира по следния начин:

```
template <class arg1, class arg2, …>
class име_на_клас
{
     декларации
};
```

Параметричният тип "аргумент" се замества в декларациите. Какъвто и аргумент (**arg**) да се появи в декларациите, **arg** се заменя от действителен аргумент (като **int, long, char ***) при използване на шаблона. За разлика от параметризираните функции при използване на параметризирани класове се използва специален синтаксис. При деклариране на обекти на параметризирания клас трябва да се посочат конкретните значения на параметризираните типове. Използва се следния ситаксис:

```
име_на_клас <конкретен_тип> име_на_обект;
```

5. Задачи за изпълнение

Задача 1:

Да се разработи шаблонна функция **maxFun** със следната декларация:

```
template <class T>
T maxFun( T a, T b)
```

Функцията връща като резултат по-големия от двата аргумента.

Да се разработи функцията **main**, като се предвидят подходящи оператори за тестване на създадената функция с различни типове данни.

Да се създаде проект с име `EX91`.

Задача 2:

Да се разработи шаблонна функция **mySwap** със следната декларация:

```
template <class T>
void mySwap( T &a, T &b)
```

Функцията разменя местата на двата аргумента.

Да се разработи функцията **main**, като се предвидят подходящи оператори за тестване на създадената шаблонна функция с различни типове данни. Шаблонната функция да се тества и с обекти от класа **ComplexNum**. Декларацията на класа **ComplexNum** е представена във файла `H:\SKELET\complex.h`. Дефинициите на методите на класа **ComplexNum** са представени във файла `H:\SKELET\complex.cpp`.

Да се създаде проект с име `EX92`, който да съдържа функцията **main,** декларацията и дефинициите на методите на класа **ComplexNum**.

Задача 3:

Да се разработи шаблонна функция **printArray** със следната декларация:

```
template <class T>
void printArray( T *array, int count)
```

Функцията разпечатва масива **array**, елементите, на който са от тип **T**, а броят им е **count**.

Да се разработи функцията **main**, като се предвидят подходящи оператори за тестване на създадената шаблонна функция с различни типове данни. Шаблонната функция да се тества и с масив от обекти от класа **ComplexNum**. Декларацията на класа **ComplexNum** е представена във файла H:\SKELET\complex.h. Дефинициите на методите на класа **ComplexNum** са представени във файла H:\SKELET\complex.cpp. За класа **ComplexNum** операцията **<<** трябва да бъде предефинирана (по-добре – като приятелска).

Да се създаде проект с име EX93, който да съдържа функцията **main,** декларацията и дефинициите на методите на класа **ComplexNum**.

Задача 4:

Да се продължи примерът от **задача 3**, като се добави нова шаблонна функция **searchArray** със следната декларация:

```
template <class T>
void searchArray( T *array, int count, T key)
```

Функцията претърсва масива **array**, елементите, на който са от тип **T**, а броят им е **count**, за стойността **key**.

Към функцията **main** да се добавят подходящи оператори за тестване на създадената шаблонна функция с различни типове данни. Шаблонната функция да се тества и с масив от обекти от класа **ComplexNum**. Декларацията на класа **ComplexNum** е представена във файла H:\SKELET\complex.h. Дефинициите на методите на класа **ComplexNum** са представени във файла H:\SKELET\complex.cpp. За класа **ComplexNum** операцията == трябва да бъде предефинирана (по-добре като член на класа).

Задача 5:

Да се разработи шаблонен клас **Pair** (двойка обекти) със следната декларация:

```
template <class T>
class Pair
{
     public:
          …….
     private:
          T a,b;
};
```

Към представената декларация да се добави инициализиращ конструктор и метод, който визуализира двата обекта.

Към функцията **main** да се добавят подходящи оператори за тестване на шаблонния клас с различни типове данни. Шаблонният клас да се тества и с обекти от класа **CStr**. Декларацията на класа **CStr** е представена във файла H:\SKELET\cstr.h. Дефинициите на методите на класа **CStr** са представени във файла H:\SKELET\cstr.cpp. За класа **CStr** операцията **<<** трябва да бъде предефинирана (по-добре – като приятелска).

Да се създаде проект с име EX95, който да съдържа функцията **main,** декларацията и дефинициите на методите на класа **CStr**.

Задача 6:

Да се разработи шаблонен клас **Store**, съхраняващ елемент от данни от произволен тип, със следната декларация:

```
template <class T>
class Store
{
public:
     Store(void);
     T getElement(void);       //retrieve item if initialized
     void putElement( T & );  //put item in storage
private:
     T item;         // item holds the data value
     int haveValue; // flag set when item is initialized
};
```

Да се разработят дефинициите на методите на шаблонния клас **Store**.

Към функцията **main** да се добавят подходящи оператори за тестване на шаблонния клас с различни типове данни. Шаблонният клас да се тества и с обекти от класа **ComplexNum**. Декларацията на класа **ComplexNum** е представена във файла H:\SKELET\complex.h. Дефинициите на методите на класа **ComplexNum** са представени във файла H:\SKELET\complex.cpp. За класа **ComplexNum** операцията **<<** трябва да бъде предефинирана (по-добре – като приятелска).

Да се създаде проект с име EX96, който да съдържа функцията **main,** декларацията и дефинициите на методите на класа **ComplexNum**.