Προγραμματισμός Υπολογιστών C++

Πρότυπες Κλάσεις (templates)

Πρότυπες Κλάσεις

- Όπως οι πρότυπες συναρτήσεις, έτσι και οι **πρότυπες κλάσεις** παρέχουν τη δυνατότητα προγραμματισμού με **γενικό** τρόπο
- Με τα πρότυπα κλάσεων (class templates) μπορούμε να ορίσουμε κλάσεις γενικής χρήσης, δηλαδή, κλάσεις στις οποίες μπορούμε να μεταβιβάσουμε **διαφορετικούς τύπους δεδομένων**, χωρίς να χρειαστεί να ξαναγράψουμε τον κώδικά τους
- Για παράδειγμα, θεωρήστε ότι έχουμε υλοποιήσει μία στοίβα (π.χ. `Lifo`) όπου ο τύπος των στοιχείων που αποθηκεύονται σε αυτήν είναι `int`. Αν θέλουμε να αποθηκεύσουμε κάποιον άλλο τύπο (π.χ. `float`) πρέπει να αλλάξουμε όλες τις εμφανίσεις του τύπου `int` με τον νέο τύπο και να μεταγλωττίσουμε πάλι το πρόγραμμα
- Δηλαδή, δεν μπορούμε να έχουμε στο ίδιο πρόγραμμα δύο `Lifo` στοίβες, η μία με `int` στοιχεία και η άλλη με `float` στοιχεία. Θα πρέπει να δημιουργήσουμε μία νέα στοίβα (π.χ. `Lifo_Float`), να αντιγράψουμε τον κώδικα της υπάρχουσας και να αλλάξουμε τον τύπο από `int` σε `float`
- Αν θέλουμε να δημιουργήσουμε μία νέα στοίβα με κάποιον άλλο τύπο δεδομένων, θα πρέπει να επαναλάβουμε την παραπάνω διαδικασία

Πρότυπες Κλάσεις

- Για να αντιμετωπίσει η C++ αυτή την αδυναμία εισάγει τις πρότυπες κλάσεις, ώστε να μην χρειαστεί να αντιγράφουμε τον ίδιο κώδικα
- Ένα παράδειγμα πρότυπης κλάσης είναι η κλάση `vector`
- Μία πρότυπη κλάση ορίζεται με **γενικό** τρόπο ώστε να μην εξαρτάται η λειτουργία της από κάποιο συγκεκριμένο τύπο
- Ο τύπος των στοιχείων που θα χειρίζεται το κάθε αντικείμενο της κλάσης μεταβιβάζεται σαν **όρισμα** στις γενικές παραμέτρους της κλάσης

Παράδειγμα

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

template <typename T> class Lifo // Πρότυπη κλάση.
{
private:
      vector<T> vec_n;
public:
      void push(const T& t);
      void pop(T& t);
      void show_all() const;
      int get_nodes() const;
};

template <typename T> void Lifo<T>::push(const T& t)
{
      vec_n.push_back(t);
}

template <typename T> void Lifo<T>::pop(T& t)
{
      int size = vec_n.size();
      if(size > 0)
      {
                  t = vec_n[size-1];
                  vec_n.erase(vec_n.end()-1);
      }
      else
                  cout << "Stack is empty\n";
}

template <typename T> void Lifo<T>::show_all() const
{
      int i, size = vec_n.size();
      if(size > 0)
      {
                  for(i = 0; i < size; i++)
                              cout << vec_n[i] << '\n';
      }
      else
                  cout << "Stack is empty\n";
}
```

```
template <typename T> int
Lifo<T>::get_nodes()  const
{
            return vec_n.size();
}

int main()
{
            int i;
            string s;

            Lifo<string> lifo_str;
            Lifo<int> lifo_int;
            lifo_str.push("One");
            lifo_str.push("Two");
            lifo_str.push("Three");
            lifo_str.pop(s);
            cout << s << '\n';
            lifo_str.show_all();

            lifo_int.push(1);
            lifo_int.push(2);
            lifo_int.push(3);
            lifo_int.pop(i);
            cout << i << '\n';
            lifo_int.show_all();
            return 0;
}
```

Πρότυπες Κλάσεις

- Όπως και με τις απλές πρότυπες συναρτήσεις, για να δηλώσουμε μία πρότυπη κλάση χρησιμοποιούμε το πρόθεμα `template <typename T>`
- Το ίδιο ισχύει όταν ορίζουμε συναρτήσεις έξω από την κλάση
- Το όρισμα `T` το οποίο θα χρησιμοποιηθεί στη δήλωση της κλάσης αντιστοιχεί σε ένα όνομα τύπου (π.χ. `int`)
- Ο τύπος `T` μπορεί να χρησιμοποιηθεί όπως ένας **συνηθισμένος** τύπος δεδομένων
- Για παράδειγμα, μπορούμε να τον χρησιμοποιήσουμε για να δηλώσουμε τον τύπο των μελών της κλάσης ή να δηλώσουμε τον τύπο επιστροφής και των παραμέτρων των συναρτήσεων μελών, είτε από μόνον του είτε συνδυασμένο σε τύπους όπως `T&` ή `T*`
- Όσον αφορά το όνομα του ορίσματος μπορείτε να επιλέξετε όποιο επιθυμείτε σύμφωνα με τους κανόνες ονοματοδοσίας. Μικρά ονόματα που αρχίζουν με το γράμμα `T` αποτελούν συνηθισμένες επιλογές
- Το πρόγραμμα εμφανίζει: `Three One Two 3 1 2`

Συγκεκριμενοποίηση Πρότυπης Κλάσης

- Το πρότυπο της κλάσης **δεν** αποτελεί ορισμό κλάσης
- Είναι απλά μία **οδηγία** για τον μεταγλωττιστή για το πώς να ορίσει την κλάση
- Η κλάση **ορίζεται** όταν απαιτείται η **συγκεκριμενοποίηση** της κλάσης για τη **δημιουργία** κάποιου **αντικειμένου**. Για παράδειγμα με την εντολή:

```
Lifo<int> lifo_int; // Είναι λάθος να γράψουμε Lifo lifo_int;
```

- ο μεταγλωττιστής **αντικαθιστά** τον τύπο `T` με `int` και ορίζεται η «ακέραια» έκδοση της κλάσης. Έτσι, ο τύπος των στοιχείων που περιέχει το `vec_n` είναι `int`
- Η κλάση `Lifo<int>` αποτελεί έμμεση συγκεκριμενοποίηση (implicit instantiation) της πρότυπης κλάσης

Συγκεκριμενοποίηση Πρότυπης Κλάσης

- Κάθε έκδοση της πρότυπης κλάσης αποτελεί μία ανεξάρτητη κλάση
- Η κλάση που παράγεται λειτουργεί όπως μία συνηθισμένη κλάση
- Παρατηρήστε ότι γράφουμε `Lifo<int>` κλάση, όχι `Lifo`, γιατί αυτή είναι η κλάση που ορίζεται για τον συγκεκριμένο τύπο. Το `lifo_int` αποτελεί αντικείμενο αυτής της έκδοσης, δηλαδή, αντικείμενο της `Lifo<int>` κλάσης. Και για να είναι ξεκάθαρο, είναι λάθος να γράψετε: `Lifo lifo_int;`
- Κάθε αντικείμενο που δηλώνετε πρέπει να αποτελεί αντικείμενο μίας συγκεκριμένης έκδοσης της πρότυπης κλάσης

Συγκεκριμενοποίηση Πρότυπης Κλάσης

- Παρόμοια, με τη δήλωση του αντικειμένου `lifo_string` ο μεταγλωττιστής αντικαθιστά τον τύπο `T` με `string` και ορίζεται η «αλφαριθμητική» έκδοση της κλάσης, δηλαδή, η `Lifo<string>`
- Η `Lifo<string>` δεν έχει καμία σχέση με την `Lifo<int>`, είναι δύο ανεξάρτητες κλάσεις
- Βλέπουμε δηλαδή ότι όταν δηλώνεται ένα αντικείμενο, ο μεταγλωττιστής αντικαθιστά τον γενικό τύπο με τον τύπο του ορίσματος και δημιουργεί την ανάλογη έκδοση της κλάσης
- Έτσι, χάρη στην πρότυπη κλάση μπορούμε να έχουμε στο ίδιο πρόγραμμα διαφορετικούς ορισμούς κλάσεων και αντίστοιχα αντικείμενα χωρίς να χρειαστεί να αντιγράψουμε κώδικα
- Δεν είναι μόνο ότι αποφεύγουμε με τις πρότυπες κλάσεις να ξαναγράψουμε παρόμοιο κώδικα. Το πρόγραμμα γίνεται πιο ευέλικτο, συμπαγές, κατανοητό και πιο εύκολο να συντηρηθεί

Χρήσεις Πρότυπης Κλάσης

- Μία πρότυπη κλάση μπορεί να χρησιμοποιηθεί όπως μία συνηθισμένη κλάση. Για παράδειγμα, μπορεί να αποτελέσει μέλος μίας άλλης κλάσης, να χρησιμοποιηθεί σαν βασική κλάση σε μία σχέση κληρονομικότητας ή να αποτελέσει τον τύπο ορίσματος στην παραγωγή κάποιας άλλης πρότυπης κλάσης. Ας δούμε ένα παράδειγμα κληρονομικότητας:

```
#include <iostream>

template <typename T> class A
{
private:
    T m;
public:
    A() {m = 10;}
    void show() const {std::cout << m << '\n';}
};

template <typename T> class B : public A<T>
{
};

int main()
{
    B<int> b;
    b.show();
    return 0;
}
```

Χρήσεις Πρότυπης Κλάσης

- Η δημιουργία του `b` αντικειμένου προκαλεί τον ορισμό της κλάσης `A<int>`, την κλήση του εξ'ορισμού κατασκευαστή και τη δημιουργία του `A` υπο-αντικειμένου που περιέχεται στο `b` αντικείμενο. Στη συνέχεια, το πρόγραμμα καλεί την `show()` της βασικής κλάσης και εμφανίζει `10`
- Στο επόμενο παράδειγμα η πρότυπη κλάση `A` αποτελεί μέλος της `C`:

```
template <typename T> class C
{
        ...
        A<T> a;
};
```

Χρήσεις Πρότυπης Κλάσης

- Στο επόμενο παράδειγμα, ένα στιγμιότυπο της προτύπης κλάσης χρησιμοποιείται ως μέλος:

```
template <typename T> class A
{
    ...
public:
    T m;
};

class C
{
    ...
public:
    A<int> a;
};

int main()
{
    C c;
    c.a.m = 10;
    ...
}
```

Χρήσεις Πρότυπης Κλάσης

- Ας δούμε και ένα παράδειγμα, όπου ο τύπος της πρότυπης κλάσης χρησιμοποιείται σαν όρισμα στη δημιουργία μίας άλλης:

```
template <typename T> class A
{
        ...
        T m;
};

A<Lifo<int>> a;
```

Το m είναι αντικείμενο της κλάσης Lifo<int>

- Επίσης, μπορούμε να δηλώσουμε ένα δείκτη σε κάποια συγκεκριμένη έκδοση της πρότυπης κλάσης και να δεσμεύσουμε μνήμη για αντίστοιχο αντικείμενο. Για παράδειγμα:

```
Lifo<int> *p;
p = new Lifo<int>; // Δημιουργία αντικειμένου
```