Wrocław 2020.10.02

Autor: Michał Przewoźniczek

Techniki Efektywnego Programowania – zadanie 3
Zależności pomiędzy klasami i obiektami, obsługa drzew

Pomiędzy poszczególnymi obiektami w programie mogą zachodzić różne zalezności. Jedną z nich jest tzw. agregacja (związek całość-część), która może być *silna* lub *słaba*. W przypadku silnej agregacji czas życia części obiektu zawiera się w czasie życia całości. Zwykle, kiedy zachodzi agragacja silna części są niszczone wraz z agregatem. W przypadku agreagcji słabej, czas życia części i całości są niezależne. Część może „przeżyć” całość na przykład wtedy, gdy części są współdzielone przez różne agregaty.

Jedną z koncepcji, wykorzystywaną w programowaniu są tzw. drzewa. Są one budowane z wielu obiektów zwanych węzłami. Drzewo jest szczególnym przypadkiem grafu. Zastosowanie drzew w programowaniu może być różne. Na przykład, drzewa mogą być przydatne w srotowaniu, wyszukiwaniu informacji, przetwarzaniu wyrażeń algebraicznych, analize języka naturalnego itp.

Niniejsze zadanie ma na celu przećwiczenie sposobu konstrukcji drzew na prostym przykładzie i wypracowanie/poprawienie umiejętności użycia (przetwarzania drzew).

Zadanie
UWAGI:

1. **Pisząc własny program można użyć innego nazewnictwa niż to przedstawione w treści zadania i w przykładach. Należy jednak użyć jakiejś spójnej konwencji kodowania, zgodnie z wymaganiami kursu.**
2. **Nie wolno używać wyjątków (jest to jedynie przypomnienie, wynika to wprost z zasad kursu).**
3. **Wolno używać wyłącznie komend ze standardu C++98**

1. Oprogramuj klasę `CNodeStatic`, której obiekty będą reprezentować węzły w drzewie.

Wymagania względem klasy `CNodeStatic`, przykładowy interfejs:

```
class  CNodeStatic
{
public:
      CNodeStatic() {i_val = 0; pc_parent_node = NULL;};
      ~CNodeStatic();

      void  vSetValue(int  iNewVal) {i_val = iNewVal;};

      int  iGetChildrenNumber() {return(v_children.size());};
      void  vAddNewChild();
      CNodeStatic  *pcGetChild(int  iChildOffset);

      void  vPrint() {cout << " " << i_val;};
      void  vPrintAllBelow();
private:
      vector<CNodeStatic>  v_children;
      CNodeStatic  *pc_parent_node;
      int  i_val;
}//class  CNodeStatic
```

- Metoda `vAddNewChild()`, dodaje nowe dziecko do wektora dzieci danego węzła
- Metoda `pcGetChild(int  iChildOffset)`, zwraca dziecko o zadanym offsecie, jeśli offset jest błędny to zwraca NULL
- Metoda `vPrintAllBelow()`, wykonuje metodę `vPrint()`, dla danego węzła, oraz `vPrintAllBelow()` dla węzłów podrzędnych (dzieci), co prowadzi do wypisania wszystkich wartości `i_val`, węzła i wszystkich dzieci w dół drzewa

2. Używając klasy `CNodeStatic`, zbuduj drzewo o kilku poziomach i przetestuj jego działanie. Na przykład możesz zbudować drzewo:

```
void  v_tree_test()
{
      CNodeStatic  c_root;

      c_root.vAddNewChild();
      c_root.vAddNewChild();

      c_root.pcGetChild(0)->vSetValue(1);
      c_root.pcGetChild(1)->vSetValue(2);

      c_root.pcGetChild(0)->vAddNewChild();
      c_root.pcGetChild(0)->vAddNewChild();

      c_root.pcGetChild(0)->pcGetChild(0)->vSetValue(11);
      c_root.pcGetChild(0)->pcGetChild(1)->vSetValue(12);


      c_root.pcGetChild(1)->vAddNewChild();
      c_root.pcGetChild(1)->vAddNewChild();

      c_root.pcGetChild(1)->pcGetChild(0)->vSetValue(21);
      c_root.pcGetChild(1)->pcGetChild(1)->vSetValue(22);
}//void  v_tree_test()
```

Zastanów się, jak powinien wyglądać destruktor, żeby wszystkie węzły zostały usunięte.

3. Dla klasy `CNodeStatic`, napisz metodę `vPrintUp()`, która dla dowolnego węzła wypisze wartości wszystkich jego rodziców, od wybranego węzła aż do korzenia. Na przykład dla drzewa z przykładu powyżej poniższa linijka:

```
c_root.pcGetChild(0)->pcGetChild(1)->vPrintUp();
```

wypisze: 12 1 0

Zastanów się jakich dane są potrzebne w klasie `CNodeStatic`, żeby wykonać metodę, opisaną powyżej.

4. Węzeł nie powinien być klasą, z którą bezpośrednio komunikuje się program. Opakuj drzewo w klasę `CTreeStatic`.

```
class  CTreeStatic
{
public:
      CTreeStatic();
      ~CTreeStatic();

      CNodeStatic  *pcGetRoot()  {return(&c_root);}
      void  vPrintTree();
private:
      CNodeStatic  c_root;
}//class  CTreeStatic
```

Meotda `vPrintTree` wypisuje na ekran wartości wszystkich węzłów.
Zastanów się, czy klasy `CTreeStatic` i `CNodeStatic` powinny być w tych samych, czy w oddzielnych plikach źródłowych.

5. Napisz klasy `CNodeDynamic` i `CTreeDynamic`, które zamiast przechowywać podrzędne węzły statycznie będą przechowywać dynamiczne wskaźniki.

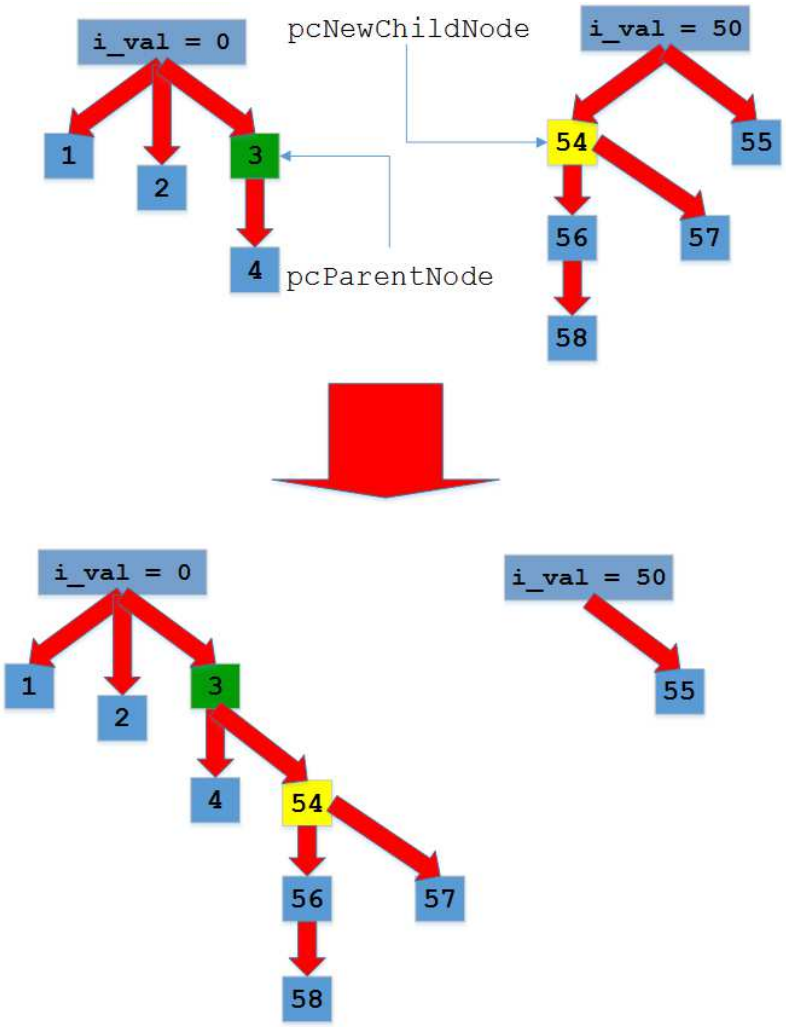
```
class  CNodeDynamic
{
public:
      CNodeDynamic() {i_val = 0; pc_parent_node = NULL;};
      ~CNodeDynamic();
      void  vSetValue(int  iNewVal) {i_val = iNewVal;};

      int  iGetChildrenNumber() {return(v_children.size);};
      void  vAddNewChild();
      CTreeDynamic  *pcGetChild(int  iChildOffset);

      void  vPrint() {cout << " " << i_val;};
      void  vPrintAllBelow();
private:
      vector<CNodeDynamic *>  v_children;
      CNodeDynamic *pc_parent_node;
      int  i_val;
}//class  CNodeDynamic

class  CTreeDynamic
{
public:
      CTreeDynamic();
      ~CTreeDynamic();

      CNodeDynamic *pcGetRoot()  {return(pc_root);}
      void  vPrintTree();
private:
      CNodeDynamic *pc_root;
}//class  CTreeDynamic
```

Czy destruktory klas `CTreeDynamic`, `CNodeDynamic` oraz `CTreeStatic`, `CNodeStatic`, powinny się czymś różnić?

6. Napisz następującą metodę dla klasy obsługującej całe drzewo.

```
bool  bMoveSubtree(CNode *pcParentNode, CNode *pcNewChildNode);
```

Metoda `bMoveSubtree`, jako parametr przyjmuje dwa węzły, **z dwóch różnych drzew**. Jeden węzeł z drzewa, dla którego została wywołana, a drugi z obcego drzewa. Celem działania metody jest usunięcie poddrzewa węzła `pcNewChildNode`, z drzewa, w którym był do tej pory i wstawienie go jako dziecka drzewa, gdzie ma się teraz znaleźć. Jest to przedstawione na Rys. 1.



Rys. 1 Przykład przenoszenia poddrzewa pomiędzy drzewami

Czy metodę `bMoveSubtree` łatwo będzie wykonać dla klas `CTreeDynamic`, `CNodeDynamic` czy dla klas `CTreeStatic`, `CNodeStatic`?
Zastanów się, czy przechowywanie list obiektów alaokowanych statycznie ma więcej zalet czy wad.

Zalecana literatura

Wykład

Materiały możliwe do znalezienia w Internecie