Obsah

Uživatelská dokumentace

Program je určen na vytváření a provádění operací s datovou strukturou AVL strom jako s objektem třídy. AVL strom je datová struktura pro uchovávání údajů a jejich vyhledávání. Pracuje v logaritmicky omezeném čase. Jedná se o samovyvažující se binární vyhledávací strom.

Vlastnosti AVL-stromu:

- V levém podstromu vrcholu jsou pouze vrcholy s menší hodnotou klíče.
- V pravém podstromu vrcholu jsou pouze vrcholy s větší hodnotou klíče.
- Délka nejdelší větve levého a pravého podstromu každého uzlu se liší nejvýše o 1.

Knihovny a hlavičkový soubory:

iostream	hlavičkový soubor s třídami, funkcemi a proměnnými pro organizaci I/O v programovacím jazyce C++.	https://learn.microsoft.com/en-us/cpp/standard-library/iostream?view=msvc-170
stdio.h	hlavičkový soubor standardní knihovny jazyka C, obsahující definice maker, konstanty a deklarace funkcí a typů používaných pro různé standardní vstupní a výstupní operace	https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/stdio.h.html
stdlib.h	hlavičkový soubor standardní knihovny jazyka C, který obsahuje funkce, které se zabývají alokací paměti, řízením provádění program a konverzí typů	https://pubs.opengroup.org/onlinepubs/009695399/basedefs/stdlib.h.html
cmath	hlavičkový soubor <cmath> deklaruje sadu funkcí pro výpočet běžných matematických operací a transformací	https://cplusplus.com/reference/cmath/
Linked_list.h	hlavičkový soubor pro prací s datovou strukturou lineární spojový seznam	

Jak na to?:

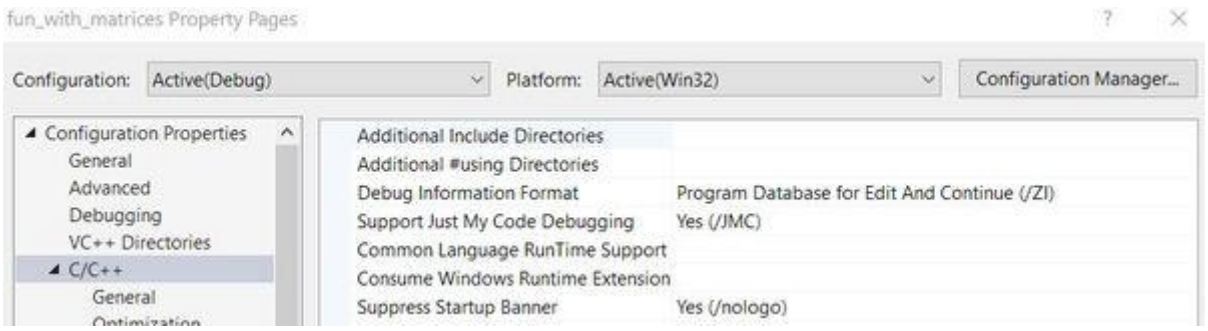
Step 1: Otevřete Visual Studio.

Step 2: Vytvořite C++ project.

Step 3: Napište tento kod do new Source file

```
#include <iostream>
#include <AVL>
```

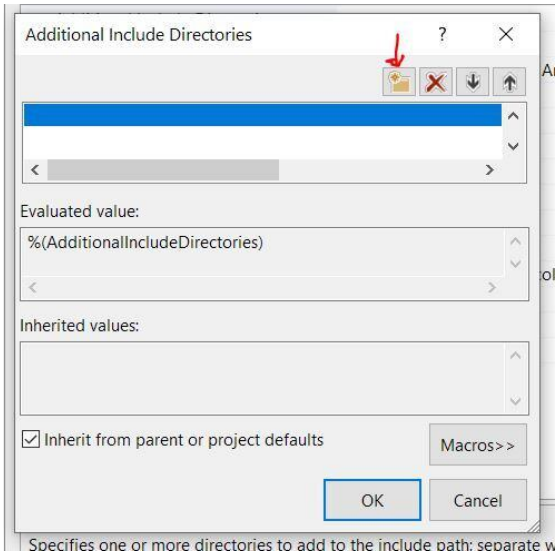
Step 4: Zmačkněte Project na hoře a jděte do Additional Include Directories.



Step 5. Zmačkněte vedle “Additional Include Directories”.

Step 6: Zmačkněte <...Edit>.

Step 7: Zmačkněte Add New Line icon.



Step 8: Přidejte direktorii knihovny AVL

Step 9: Zmačkněte OK.

Step 10: Zmačkněte Apply a pak OK.

Jak vytvořit objekt třídy?:

- Konstruktory:
- Bez parametrů:

```
AVL_node() {
    data = 0;
    height = 1;
    left = NULL;
    right = NULL;
};
```

modifikátor: public

- S parametry:

```
AVL_node(int d) {
    data = d;
    height = 1;
    left = NULL;
    right = NULL;
};
```

parametry: d: int
hodnota atributu *data*

modifikátor: public

Příklad incializaci:

Vytváření prazdného vrcholu AVL stromu:

```
AVL_node vrchol;
```

Vytváření vrcholu AVL stromu s hodnotou 6:

```
AVL_node vrchol(6);
vrchol.show();
```

Output:

```
(6).(0)
```

Návod k vytváření AVL stromu:

Příklad:

Vytváření AVL stromu

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(-11);
    vrchol.insert(-11);
    vrchol.insert(-11);
    vrchol.insert(0);
    vrchol.insert(-12);
    vrchol.insert(-1);
    vrchol.insert(10);
    vrchol.insert(2);
    vrchol.insert(-5);
    vrchol.del(2);

    vrchol.show();
}
```

Output:

```
    ,--(10).(1)
    |    |
    |    '--(1).(0)
    |
(0).(1)
    |
    |    ,--(-1).(1)
    |    |    |
    |    |    '--(-5).(0)
    |    |
    '--(-11).(-1)
         |
         '--(-12).(0)
```

Na obrazku vyše vidíme strukturu vytvořeného stromu. První hodnota ve závorkach je hodnotou vrcholu, druha je balance odpovídaícího vrcholu.

Při vytváření prvku třídy musíte předat hodnotu **x** vytvářeného vrcholu. Pokud vrcholu nepředána žádná hodnota, pak bude hodnota vrcholu nastavena na 0. Pokud uzel nemá žádné z potomků, pak budou se hodnoty synů rovnat NULL. Speciálně v listech stromu jsou oba ukazatele nastaveny na NULL.

Pokud opakovavaně přidáváte existující hodnotu, tak strom zustane nepozměnený (viz void insert(int a)). Stejně pokud mazáte neexistujicí hodnotu, tak strom zustane nepozměnený (viz void del(int a)).

```
void insert(int a)
```

Přidava k AVL stromu nový vrchol s zadanou hodnotou **a,** pokud vrchol se zádanou hodnotou **a** neexistuje.

parametry: a: int
hodnota, přidáváného vrcholu

modifikátor: public

Příklad:

Vytváření AVL stromu s hodnotami 1, 2, 3

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(2);
    vrchol.insert(3);

    vrchol.show(); //vypíše do konzole strukturu stromu
}
```

Output:

```
    ,--(3)
    |
 (2)
    |
    '--(1)
```

```
void del(int a)
```

Vypouští z AVL stromu vrchol s zadanou hodnotou **a**, pokud vrchol se zádanou hodnotou **a** existuje.

parametry: a: int
hodnota vypouštěného vrcholu
modifikátor: public

Příklad:

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(2);
    vrchol.insert(3);

    vrchol.del(2);

    vrchol.show();
}
```

Output:

```
(3)
   |
   '--(1)
```

```
int find(int a)
```

Funkce najde hodnotu **a** ve stromu. Pokud hodnota byla nelezena, vratí 1 a vypíše do konzole «Hodnota a byla nalezena». Jinak, vratí 0 a vypíše «Hodnota a nebyla nalezena».

parametry: a : int
hledaná hodnota
modifikátor: public
vrátí: int
1 – hodnota byla nalezena
0 – hodnota nebyla nalezena

Příklad:

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(2);
    vrchol.insert(3);

    int a = vrchol.find(2);
    cout << a << endl;
    vrchol.find(7);
}
```

Output:

```
Hodnota 2 byla nalezena
1
Hodnota 7 nebyla nalezena
```

- `int findmin()`

Funkce najde hodnotu listu s minimální hodnotou.
modifikátor: public
vrátí: int
minimální hodnota stromu

Příklad:

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(2);
    vrchol.insert(3);

    cout << vrchol.findmin() << endl;
}
```

Output:

```
1
```

- `int findmax()`

Funkce najde hodnotu listu s maximální hodnotou.
modifikátor: public
vrátí: int
maximální hodnota stromu

Příklad:

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(2);
    vrchol.insert(3);

    cout << vrchol.findmax() << endl;
}
```

Output:

```
3
```

- `int next()`

Je to iterator, který vrací postupně hodnoty od nejmenší k největší. Pokud dojdou hodnoty, vratí stále největší ale k tomu navíc vypíše do konzole «`Další hodnoty nejsou`».

modifikátor: public

vrátí: int

další nejmenší hodnota

Příklad:

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(2);
    vrchol.insert(3);

    cout << vrchol.next() << endl;
    cout << vrchol.next() << endl;
    cout << vrchol.next() << endl;
    cout << vrchol.next() << endl;
}
```

Output:

```
1
2
3
Dalsi hodnoty nejsou
3
```

```
void show(bool fromleft = 0, bool prev = 0,
linked_list* prev_state = NULL)
```

Funkce, vypisující do konzole strukturu AVL stromu.

parametry: `fromleft: bool`
proměnna pomocna k určení jestli funkce byla rekurzivně zavolána z levé větví.
0 - funkce byla zavolána z levé větví
1 - funkce byla zavolána z pravé větví
`prev: bool`
proměnna pomocna k určení parametru fromleft
`prev_state: linked_list*`
parametr, ve kterém je uložen předchozí řadek, vitisknutý v konzoli
modifikátor: public

Příklad:

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(2);
    vrchol.insert(3);

    vrchol.show();

}
```

Output:

```
    ,--(3)
    |
(2)
    |
    '--(1)
```

Závěr:

Tato knihovna umožňuje vytvářet AVL stromy a jednoduše s nimi provádět základní operace: vypsat hodnoty uzlů AVL, přidat nový uzel s zadanou hodnotou, najit v AVL zadanou hodnotu, najit maximální nebo minimální hodnotu v AVL, vypustit uzel AVL s zadanou hodnotou. Také knihovna umožnuje vypisovat do konzole jakou momentalně strukturu má AVL strom, což je užitečný nástroj, umožnující snadno nahlednout fungování AVL-stromu (jak ve ktere situaci vypadá, kde je jaká balance a jaké operace se prováději).