class **Stack**

1. Tworzenie stosu

Konstruktory domyślne

- **Konstruktor domyślny** `Stack stos` **- tworzy stos i kładzie na niego wartość 0**
- **Konstruktor z parametrem** `Stack stos(2)` **- tworzy stos i kładzie na stos podaną wartość int**

Klasa posiada również konstruktor kopiujący

- **Konstruktor kopiujący** `Stack stos(Stack innyStos)` **- tworzy kopię podanego w argumencie stosu**

2. Wypełnianie i pobieranie elementów ze stosu

Za pomocą metody

- ***Metoda*** **Put(**`int`**)** `stos.Put(2)` **- kładzie na stos podany w parametrze** `int`
- ***Metoda*** **Get()** `stos.Get()` **- zwraca pobrany z góry stosu** `int`

Za pomocą przeciążonych operatorów

- ***Operator*** **+** `stos + 2` **- kładzie na stos dodawany** `int`
- ***Operator*** **-** `-stos` **- zwraca pobrany z góry stosu** `int`
- ***Operator*** **+=** `stos += innyStos` **- kładzie na stos cała zawartość innego stosu klasy** `Stack`

> w przypadku gdy nastąpi próba pobrania elementu z pustego stosu zostanie wyświetlony komunikat "Underflow.." i pobrany zostanie - `int` równy `0`

3. Wyświetlanie stosu

Wyświetlanie aktualnego stosu możliwe za pomocą przeciążonego operatora iostream

- ***Operator*** **cout <<** `cout << stos` **- wypisuje stos w konsoli**

4. Pozostałe metody

Stan stosu

- ***Statyczna metoda*** **State()** `stos.State()` **- zwraca informacje o stanie stosu w postaci enum (**`empty` **=** `false`**,** `contains` **=** `true`**)**

Rozmiar stosu

- ***Statyczna metoda*** **Size()** `stos.Size()` **- zwraca** `int` **przedstawiający ilość elementów zapisanych na stosie**

Przykłady

Tworzymy stos o nazwie `stos1` wypełniony `int`-em o wartości `13`:

```
    Stack stos1(13);
```

Za pomocą **metody** oraz **operatora** kładziemy na stosie `6` a następnie `14` oraz wyświetlamy całość stosu:

```
    stos1.Put(6);
    stos1 + 14;
    cout << stos1;
```

w konsoli otrzymamy:

```
KONSOLA
\/--Stack--
|  14
|  6
|  13
\/--------
```

Zdejmiemy teraz ze stosu 2 górne wartości, korzystając z **metody** oraz **operatora** a nastęonie wyświetlmy stos:

```
    cout << stos1.Get() << endl;
    cout << -stos1 << endl;
    cout << stos1;
```

```
KONSOLA
14
```

```
6
\/--Stack--
|  13
\/--------
```

Spróbujmy teraz pobrać ze stosu więcej elementów niż się w nim znajduje:

```
    cout << -stos1 << endl;
    cout << -stos1 << endl;
    cout << -stos1 << endl;
```

```
KONSOLA
13
underflow..
0
underflow..
0
```

Sprawdźmy stan naszego stosu za pomocą *metody* `State()` oraz ilość elementów które zawiera za pomocą *metody* `Size()`:

```
    cout << "stan stosu: " << stos1.State() << endl;
    cout << "rozmiar stosu: " << stos1.Size() << endl;
```

```
KONSOLA
stan stosu: 0
rozmiar stosu: 0
```

Dodajmy na stos kilka nowych elementów i sprawdźmy ponownie:

```
    stos1 + 2;
    stos1 + 15;
    stos1 + 4;
    stos1 + 86;
    cout << "stan stosu: " << stos1.State() << endl;
    cout << "rozmiar stosu: " << stos1.Size() << endl;
```

```
KONSOLA
stan stosu: 1
rozmiar stosu: 4
```

Na koniec spróbujmy stworzyć kopię naszego stosu a następnie położyć ją na górę stosu `stos1`:

```
    Stack stos2(stos1);
    stos1 += stos2;
    cout << stos1;
```

```
KONSOLA
\/--Stack--
|  86
|  4
|  15
|  2
|  86
|  4
|  15
|  2
\/--------
```

2022 r. projekt z programowania komputerów, Zuzanna Chrabańska, Paweł Czaja, Igor Ignacek.