Lab1

5) Utwórz nowy projekt i napisz program, który będzie:

   a) Wyprowadzał na monitor przywitanie: nazwisko, imię oraz numer grupy.
   b) Wczytywał się rok urodzenia, na podstawie, którego obliczy wiek i wypisze go na ekran.
   c) Wczytywał liczbę int, double oraz wiersz tekstowy składający się z kilku słów rozdzielonych spacjami
   d) Wyprowadzał zmienne z pkt *c* na monitor

   Do wprowadzania i wypisywania danych należy użyć operatorów C++ >> oraz <<

6) Dodaj do programu tablicę typu double i wypełnij kilkoma wartościami np.: 10, 20, … Następnie wypisz na ekranie dane z tabeli w formacie:

   NAGŁÓWEK

   Pozycja Wartość
   1 10.0
   2 20.0
   .....

7) Zmodyfikuj program tak, aby pytał użytkownika o wprowadzenie 3 wartości całkowitych, a następnie wyświetlał je w kolejności, w której zostały wprowadzone oraz w kolejności odwrotnej.

8) Utwórz nowy projekt i napisz program „Kalkulator”, który będzie posiadał podstawowe operacje arytmetyczne (dodawanie, odejmowanie, mnożenie i dzielenie). Użytkownik wybiera, jaką operację chce wykonać, a następnie podaje 2 liczby, na których wybrana operacja ma się wykonać. W wyniku program powinien zwrócić wynik obliczeń.

Lab2

2. Stwórz klasę *my_vol*, której zadaniem jest obliczenie objętości prostopadłościanu $V = abc$, gdzie a, b, c – są odpowiednimi wymiary bryły. Wartości *a, b, c* (typu double, jako składowe prywatne klasy) muszą być przekazywane do konstruktora tej klasy. Dane wprowadzane przez użytkownika z monitora. Klasa musi zawierać:
   a. funkcję obliczania objętości *V*.
   b. funkcję wyświetlania wyniku na monitor.

3. Zapoznaj się z działaniem wzorcowego przykładu *Complex_class* (dostępny w materiałach na stronie www). Stwórz klasę *Triangle*, w której trzy punkty *A, B, C* są wierzchołkami. Klasa *Triangle* powinna zawierać konstruktor sparametryzowany oraz funkcję wyświetlania współrzędnych każdego wierzchołka na monitorze. Stwórz obiekt typu *Triangle* i wyświetlić go na monitorze.

4. Zmienić program tak, aby maksymalną ilość elementów stosu przekazywać

konstruktorowi klasy. W konstruktorze klasy użyć dynamiczną alokacje pamięci. Wprowadzić destruktor, który zwalnia pamięć.

5. Stwórz klasę *time_day*, która przy tworzeniu pobiera czas CPU za pomocą funkcji *GetTickCount()* (funkcja zwraca typ DWORD – unsigned long) (w tym celu dołącz plik nagłówkowy "*windows.h*" -> patrz MSDN). W tym celu wykorzystaj konstruktor sparametryzowany. Klasa musi zawierać funkcję, która wypisze datę na monitorze. Destruktor klasy musi ponownie wywołać funkcje *GetTickCount()* i obliczyć czas, który upłynął od momentu utworzenia obiektu klasy do jego zniszczenia i wypisać ten czas na monitorze. W celu zatrzymania czasu wykonywania programu o 2 sekundy użyj funkcji platformy SDK *Sleep(2000)* (*#include "windows.h"*). Użyj obiektów klasy jako globalne, lokalne oraz typu static (lokalne). Wypróbuj działanie dla globalnego i lokalnego obiektu klasy. Prześledź miejsca w programie, w których odbywa sie wywołanie konstruktorów i destruktorów dla tych obiektów.

Lab3

1. Stworzyć nowy projekt. Zdefiniować klasę kolo, która powinna zawierać:
    a. pola prywatne:
        i. promień koła (typ float)
        ii. kolor wypełnienia (typ int)
    b. konstruktor bezparametrowy inicjujący wartość początkową pola koła (dowolnie ustaloną)
    c. konstruktor z parametrami: promień, kolor wypełnienia
    d. metody publiczne:
        i. Oblicz_Pole(..), obliczająca pole powierzchni koła
        ii. Ustaw_Promien(..), zmieniająca promień koła
        iii. Ustaw_Kolor(..), zmieniającą kolor koła
        iv. funkcję zaprzyjaźnioną Porownaj_Kola(KOLO A, KOLO B), porównującą, czy dwa koła mają taką samą powierzchnię i kolor.

Lab4

1. Stworzyć nowy projekt.
   Zdefiniuj klasę *prostokat*, która powinna zawierać:
    a. pola prywatne typu całkowitego:
        i. długości (długość i szerokość) boków prostokąta
        ii. numer identyfikacyjny
    b. konstruktor bezparametrowy inicjujący wartość początkową pól prostokąta (dowolnie ustaloną)
    c. konstruktor z parametrami długość, szerokość i numer identyfikacyjny prostokąta
    d. metody publiczne:
        i. *Get_Dlugosc(..)* , zwracającą długość prostokąta
        ii. *Get _Szerokosc(..)*, zwracającą szerokość prostokąta

iii. Zdefiniuj funkcję zaprzyjaźnioną *Czy_Kwadrat(PROSTOKAT A)*, porównującą boki prostokąta (sprawdzającą czy jest kwadratem).

Lab5

a) Stworzyć nowy projekt. Dodać do projektu plik node_coord.h, w którym umieścić deklaracje klasy NODE_COORD:

```
class NODE_COORD
{
        double *pcoord;     //pcoord[0] - x, pcoord[1] - y
public:
        NODE_COORD() : pcoord(NULL) {}      //konstruktor domyślny
        NODE_COORD(double x, double y);     //konstruktor sparametryzowany
        ~NODE_COORD();                      //destruktor
        void disp();                        //wyświetla na monitorze x, y
private:
        void crash();               //obsługuje błąd alokowania pamięci
};
```

b) W pliku *.cpp dodać implementacje metod klasy NODE_COORD

```
NODE_COORD::NODE_COORD(double x, double y)
{
        try
        {
                pcoord = new double [2];
                pcoord[0] = x; pcoord[1] =
                y;
        }
            catch(bad_alloc){
                crash();
        }
}
void NODE_COORD::crash()
{
        cout << "memory allocation error\n";
        system("pause");
        exit(1);
}
NODE_COORD::~NODE_COORD()
{
        if(pcoord)
        {
                delete [] pcoord;
                pcoord = NULL;
        }
}
```

c) W funkcji *main* tworzymy testowe obiekty klasy:

```
int _tmain(int argc, _TCHAR* argv[])
{
        NODE_COORD A(2,
```

```
        3); NODE_COORD
        B = A;

        NODE_COORD C;
        NODE_COORD D = C;

        system("pause
        "); return 0;
}
```

d) Uruchomić program, wyjaśnić przyczynę niepowodzenia i poprawić kod tak, żeby działał poprawnie.

e) Dodać do zadania klasę *Triangle* (pliki *triangle*.h, *triangle*.cpp), która zawiera trzy wierzchołki typu NODE_COORD *vert_A*, NODE_COORD *vert_B*, NODE_COORD *vert_C* oraz wiersz tekstowy *str[128],* przeznaczony do przechowywania nazwy trójkąta (na przykład, „trojkąt 1"). Do wprowadzania danych wykorzystać konstruktor sparametryzowany. Unikając tym samym korzystania z operatora przypisania. Klasa *Triangle* powinna zawierać metodę *disp()* wyświetlającą współrzędne każdego z wierzchołków, wywołując metodę *disp()* z klasy wierzchołków (*NODE_COORD*). Deklaracje klasy *Triangle* umieścić w pliku *triangle*.h, a implementacje metod klasy – w pliku *triangle*.cpp. Funkcja *main* wygląda następująco:

```
int _tmain(int argc, _TCHAR* argv[])
{
        NODE_COORD A(2, 3);
        NODE_COORD B = A;

        NODE_COORD C;
        NODE_COORD D = C;

        NODE_COORD AA(2, 3), BB(3,4), CC(0, 0);
        Triangle tr(AA, BB, CC, "trojkat 1");
        tr.disp();

        system("pause");
        return 0;
```

f) Dodać do klasy *Triangle* metodę *double distance*(*int First, int Second*), która liczy odległość pomiędzy wierzchołkami o numerze *First* i *Second*:

g) Zmienna *pcoord* klasy NODE_COORD powinna pozostać jako *private*, dlatego w pliku *node_coord*.cpp należy dodać funkcję *double distance*(NODE_COORD A, NODE_COORD B) i *zaprzyjaźnić* ją do klasy NODE_COORD (*friend double distance*(NODE_COORD A, NODE_COORD B);). Metoda *distance* klasy *Triangle* powinna korzystać z funkcji *double distance*(NODE_COORD A, NODE_COORD B).

```
int _tmain(int argc, _TCHAR* argv[])
{
        NODE_COORD A(2, 3);
        NODE_COORD B = A;
        NODE_COORD C;
```

```
    NODE_COORD D = C;

    NODE_COORD AA(2, 3), BB(3, 4), CC(0, 0);
    Triangle tr(AA, BB, CC, "trojkat 1");
    tr.disp();
    cout << "distance between first and second nodes: " << tr.distance(1, 0) << endl;
    system("pause");
    return 0;
}
```

h) Stworzyć funkcję o nazwie *void fun(Triangle trr)*, która przyjmuje obiekt typu *Triangle* i wywołuje metodę *disp()* do wyświetlania na monitorze. Przeciążyć tą funkcje *void fun(Triangle *trr)* i zrobić to samo, tylko korzystając ze wskaźnika do obiektu klasy *Triangle*. Funkcja *main* teraz wygląda tak:

```
int _tmain(int argc, _TCHAR* argv[])
{
    NODE_COORD A(2,
    3); NODE_COORD
    B = A;

    NODE_COORD C;
    NODE_COORD D = C;

    NODE_COORD AA(2, 3), BB(3,4), CC(0, 0);
    Triangle tr(AA, BB, CC,
    "trojkat 1"); tr.disp();

    my_fun(tr);
    my_fun(&t
    r);

    system("pause
    "); return 0;
}
```

i) Przy pomocy debuger'a wyjaśnić różnicę przy wywołaniu funkcji pierwszej i drugiej.

Lab6

a) Stwórz nowy projekt.

Skopiuj poniższy kod do swojego środowiska programistycznego, aby móc go wypróbować u siebie:

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

// Prototyp funkcji
int liczbaSekund(int godziny, int minuty, int sekundy);

// Funkcja główna int main(){
        cout << liczbaSekund(1, 10, 25) << endl; return
        0;
}

// Definicja funkcji
```

```
int liczbaSekund(int godziny, int minuty, int sekundy)
{
                    int suma = 0;
                    suma = godziny * 60 * 60;
                    suma += minuty * 60;  suma
                    += sekundy;

                    return suma;
}
```

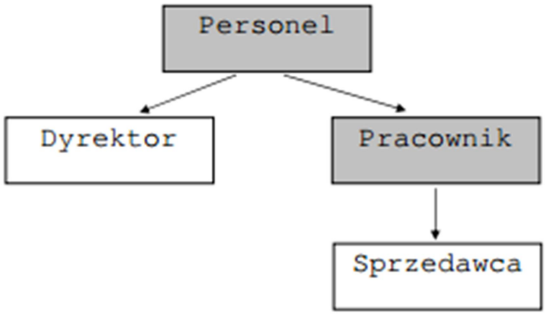
b) Wynik działania tego programu jest następujący: 4225
c) Skąd wzięła się ta liczba? 1 godzina = 3600 sekund, 10 minut = 600 sekund, 25 sekund = 25 sekund, a więc 3600 + 600 + 25 = 4225.
d) Zmodyfikuj prototyp funkcji tak, aby podanie niektórych argumentów wywołania tej funkcji było nieobowiązkowe, ponieważ np. najczęściej znana jest tylko liczba godzin.
e) W funkcji *main()* umieść wywołanie: `cout << liczbaSekund(1) << endl;`
f) Czy można przekazać do funkcji liczbaSekund() tylko liczbę godzin i sekund, bez minut?
g) Czy można sprawić, aby tylko liczba godzin była opcjonalna, a liczby minut i sekund obowiązkowe?
h) Czy można wszystkie parametry zdefiniować, jako opcjonalne?

Lab8

1. **Stwórz nowy projekt.**
2. Napisz program do zarządzania pracownikami firmy X. Założenia programu:
    a. W programie należy zapamiętać nazwisko (*nazwisko*) pracownika i numer pokoju (*biuro*) , w którym pracownik przebywa.
    b. Na podstawie numeru pokoju odpowiednia funkcja ma wyświetlić numer telefonu do pracownika (baza danych numer pokoju / telefon może być „zapisana” w kodzie programu).
    c. Firma X zatrudnia 3 rodzaje pracowników opłacanych w różny sposób:
        i. Pracownicy pracujący na akord. Wypłata pracownika (pensja) równa się liczbie przepracowanych w miesiącu godzin (*godziny*) pomnożonych przez stawkę godzinową (*stawka*).

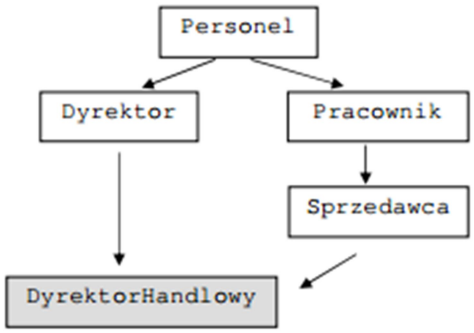
        Sprzedawcy – wypłata równa się stałej pensji (wyliczonej jako ustalona stawka razy ustalona liczba godzin – tak jak w przypadku Pracownika) + dodatek wyliczony w oparciu o procent (*procent*) zrealizowanej w danym miesiącu sprzedaży (*sprzedaż*).
        ii. Dyrektorzy – wypłata równa się stałej pensji (*pensja*) + premia (*premia*) uzależniona od liczby podległych mu pracowników (*ile_pracowników*). Uwaga: pensja i premia Dyrektora jest ustalana z góry i nie zależy od godzin, stawki, procentów jak w przypadku Pracowników i Sprzedawców zdefiniowanych wyżej.

d. Struktura dziedziczenia klas w programie zgodnie z poniższym diagramem:



e. Pola klas (wyszczególnione kursywą) jako prywatne składowe.

3. Zmodyfikuj program dodając kolejną klasę: DyrektorHandlowy (tak jak na diagramie poniżej). Dyrektor handlowy zarządza pewną ilością pracowników (cecha dziedziczona z klasy Dyrektor) oraz otrzymuje pensje i premię uzależnioną od ilości sprzedaży (cecha dziedziczona z klasy Sprzedawca). Każdy obiekt typu DyrektorHandlowy ma dostęp do składowych (publicznych) klas Dyrektor i Sprzedawca (wykorzystaj mechanizm wielodziedziczenia i dziedziczenia wirtualnego).



Lab9

1. Klasa *Figura* jest klasą bazową dla figur płaskich *Circle*, *Triangle*, *Rectangle*. Zawiera funkcje abstrakcyjne *void area()* oraz *void disp()*. Funkcja *area* jest przeznaczona dla obliczenia pola figury płaskiej, a *disp* – dla wyświetlania na monitorze tego pola.

```
class Figura
{
public:
	static size_t alloc;  //ile razy była dynamicznie alokowana pamięć dla
	//obiektów klas pochodnych
protected:
	double s;		//Pole figury płaskiej
public:
	…..................................
	//deklarowanie metod klasy
};
```

Stworzyć klasy pochodne *Circle, Triangle, Rectangle*, które reprezentują odpowiednie figury płaskie koło, trójkąt oraz prostokąt. Dane przekazać za pomocą konstruktorów

sparametryzowanych. *Przesłonić* w każdej klasie metody *area* oraz *disp* tak, żeby metoda *area* liczyła pole odpowiedniej figury i wynik przypisywała zmiennej *s*, a *disp* – wyprowadzała na monitor nazwę figury oraz jej pole.

Klasa *Rectangle* w konstruktorze klasy dynamicznie alokuje pamięć dla zmiennej *dat* i inkrementuje zmienną *alloc*. Destruktor powinien zwolnić tą pamięć i dekrementować zmienną *alloc*.

```
class Rectangle : public Figura{

    double *dat; //dat[0] - a, dat[1] - b; a, b - długości boków
public:
//metody klasy
};

//Funkcja main wykonuje kod:
int _tmain(int argc, _TCHAR* argv[]){

    Figura *ptr = NULL, *ptr_rect = NULL;
    //tworzymy obiekt Rectangle i jego wskaźnik przypisujemy do wskaźnika typu klasy
    //bazowej
    ptr_rect = new Rectangle (2, 3);
    //Tworzymy obiekty
    Circle cl(2), cll(3);
    Triangle tr(2, 4);
    srand(time(NULL));
    for(int it=0; it<10; ++it){
        int ind = rand()%4; //teraz ind przyjmuje wartości 0, 1, 2, 3 w sposób
                            //losowy
        switch(ind){
            case 0: ptr = &cl;
            break;
            case 1: ptr = &cll;
            break;
            case 2: ptr = &tr;
            break;
            case 3: ptr = ptr_rect;
            break;
        };
        //tu ptr - wskaźnik klasy bazowej - losowo wskazuje do jednego z obiektów klas
        //pochodnych. Dla jakich obiektów będą wywołane funkcji area, disp ?
        ptr->area();
        ptr->disp();
    }

    delete ptr_rect;
    ptr_rect = NULL;
    //jeśli poprawnie alokujemy i dealokujemy pamięć, alloc powinien być równy zero.

    if(Figura::alloc)
    cout << "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n" << "error: leak of memory\n";

    system("pause");
    return 0;
}
```

Lab10

1. Dane są dwa wskaźniki. Jeden z nich jest ustawiony na obiekt o nazwie szesc. Napisz instrukcję (rzutowania), która pozwoli przepisać adres ze wskaźnika wsk_c do wskaźnika wsk_norm.

```
const int szesc = 6;
const int *wsk_c =
&szesc; int *
wsk_norm;
```

Lab11

1. Napisz program wyświetlający na ekranie wizytówkę o podanej formie. Wpisz w wizytówce swoje dane. Zastosuj odpowiednie strumienie i operatory.

```
********************************
*         Jan Kowalski         *
* e-mail: j.kowalski@gmail.com *
*       tel. 123-456-789       *
********************************
```

2. Napisz program, w którym użytkownik podaje z klawiatury kwotę w złotych (PLN). Następnie pogram wyświetla informację, ile za tę kwotę można kupić innych walut (EUR, USD, GBP, CHF, CAD, DKK, JPY). Poszukaj w Internecie kursów kantorowych. Wyniki przedstaw w postaci tabeli. Przyjmij, że część całkowita wyświetlanych liczb nie będzie zawierać więcej niż 7 cyfr. Zadbaj o odpowiednie wyrównanie wyświetlanych liczb. Zastosuj odpowiednie strumienie i manipulatory. Przykładowy wynik działania programu:

```
Podaj kwote w PLN: 100

|-------------|----------|------------|
|    Kraj     |  Waluta  |   Kwota    |
|-------------|----------|------------|
| EUGiW       |   EUR    |       x.xx |
| USA         |   USD    |       x.xx |
| W. Brytania |   GBP    |       x.xx |
| Szwajcaria  |   CHF    |       x.xx |
| Kanada      |   CAD    |       x.xx |
| Dania       |   DKK    |       x.xx |
| Japonia     |   JPY    |       x.xx |
|-------------|----------|------------|
```

1. Poniżej znajduje się kod klasy LZesp z zaimplementowanymi operatorami +, = . Należy doimplementować wymienione niżej operatory, tak, aby kod dał się poprawnie skompilować i działał zgodnie z intuicją (oraz regułami matematyki).

- LZesp operator-(const LZesp &z) - odejmowanie liczb zespolonych
- LZesp operator*(const LZesp &z) - mnożenie liczb zespolonych
- bool operator==(const LZesp &z) - porównanie (stwierdzenie równości)
- bool operator!=(const LZesp &z) - porównanie (stwierdzenie różności)
- double operator[](int idx) - zwrot części rzeczywistej (jeśli idx==0) lub urojonej (jeśli idx!=0). Inaczej mówiąc: z[0] ma być tym samym, co z.getRe(), a z[1] tym samym,

coz.getIm().

- istream& operator>>(istream &i, LZesp &l) - wczytanie liczby zespolonej z klawiatury (należy wykorzystać zdefiniowany już operator >> dla typu double)

Komentarz:

Kompilator C++ nie jest w stanie przewidzieć co rozumiemy przez *l2 + l1* czy *cout << l3*. Zatem musimy zdefiniować, co w przypadku obiektów klasy LZesp oznacza dodawanie, odejmowanie, mnożenie czy przesłanie do strumienia wyjściowego Na przykład, aby określić co oznacza operator + w przypadku obiektów klasy LZesp, musimy zdefiniować odpowiednią metodę (lub funkcję) operatorową o *prototypie: LZesp operator+(const LZesp &z)*. Wówczas wyrażenie *l1 + l2* będzie oznaczało dla kompilatora *l1.operator+(l2)*. Aby możliwe było wyrażenie *l3 = l1 + l2* należy również przeciążyć operator przypisania = (za pomocą np. metody *LZesp & operator=(const LZesp &z)*).

```
#include "stdafx.h"
#include <iostream>
// aby można było użyć cin i cout

using namespace std;

class LZesp {
protected:
    double re;
    double im;

public:
    LZesp(double are=0, double aim=0){
        re = are; im = aim;
    }

    LZesp(const LZesp &z){
        re = z.getRe();
        im = z.getIm();
    }
    double getRe() const{
        return re;
    }
    double getIm() const{
        return im;
    }
    void set(double are, double aim){
        re = are; im = aim;
    }
    LZesp operator+(const LZesp &l) const{
        return LZesp(re + l.getRe(), im + l.getIm());
    }
    LZesp& operator=(const LZesp &l){
        if (&l != this) {
            re = l.getRe();
            im = l.getIm();
        }
        return *this;
    }
};
```

```
int main(void)
{
    LZesp l1(2,3), l2(3,4), l3;
    l3 = l1 + l2;
    // tak naprawdę:
    // l3.operator=( l1.operator+(l2) );

    // kod poniżej można odkomentować dopiero po implementacji
    // odpowiednich operatorów

    // cout << l1 << " + " << l2 << " = " << l3 << endl;
    //LZesp l4, l5;
    //cin >> l4;
    //cin >> l5;
    //cout << l4 << " * " << l5 << " = " << l4*l5 << endl;
    //cout << l4 << " - " << l5 << " = " << l4-l5 << endl;
    //if (l4 == l5) cout << l4 << " jest rowna " << l5 << endl;
    //else cout << l4 << " nie jest rowna " << l5 << endl;
    //if (l4 != l5) cout << l4 << " rozni sie od " << l5 << endl;
    //else cout << l4 << " nie rozni sie od " << l5 << endl;

    return 0;
}
```

Lab12

1. Pewien plik tekstowy zawiera wyniki kolokwium zapisane w następującym formacie:

```
Albert Einstein 1.5 -5 0.75
Fryderyk Chopin 20.5 20 5.25
Maria Sklodowska-Curie 50 50 50
```

W każdej linii znajduje się jedno imię, jedno nazwisko, a następnie liczby punktów za kolejne zadania. Liczba zadań ani osób nie jest z góry znana, wiadomo jednak, że liczba zadań jest dla wszystkich taka sama. Napisz program *colloquium*, który na podstawie tego pliku obliczy i wypisze całkowitą liczbę punktów zdobytych przez każdą osobę oraz średnią liczbę punktów z każdego zadania. W podanym przypadku wynikiem działania programu powinien być wydruk:

```
Albert Einstein -2.75
Fryderyk Chopin 45.75
Maria Sklodowska-Curie 150
1 24
2 21.6667
3 18.6667
```

Użycia ilu wektorów wymaga to zadanie?

Lab13

1. Napisz program, który zrealizuje następującą hierarchię klas: shape_geom – klasa bazowa, zawierająca składową prywatną typu double area. Klasy pochodne: rect, circle. Do wprowadzenia danych w klasach pochodnych wykorzystaj konstruktory sparametryzowane. Składowe klas pochodnych zadeklaruj jako private. Każda z klas pochodnych musi mieć metodę obliczenia pola dla prostokąta (rect) i kola (circle), które przekazują wynik składowej klasy bazowej area. Do wyświetlania wyniku klasa bazowa powinna udostępnić zmienną prywatną area.

2. . Dane są dwie klasy *pr1* oraz *pr2*, które wykorzystują wspólną drukarkę. W całym programie trzeba wiedzieć, kiedy drukarka jest zajęta obiektem jednej z tych klas. Należy stworzyć funkcję *inuse(),* która zwraca *true*, jeśli drukarka jest zajęta, i *false*, jeśli jest wolna. Funkcja *inuse()* ma być funkcją zaprzyjaźnioną jednocześnie do klasy *pr1* i *pr2* (szkielet programu poniżej).

```
class pr1{
    int printing; public:
    pr1() {printing = 0; }
    void set_printing(int status) { printing = status; }
};
class pr2{
    int printing; public:
    pr2() { printing = 0; }
    void set_printing(int status) { printing = status; }
};
int main(){
    pr1 ob1; pr2 ob2; int it=0;
    int key, it_used = 0;
    while(it<100){
        key = rand(); //wartoec losowa key
        zwolnic jej
        //-----------jesli drukarka jest zajeta 4 iteracji petli -
        if((it-it_used) >= 4){
            if(ob1.is_used()){
                //jesli obiekt 1 zajmuje drukarke, zwalniamy jej ob1.set_printing(0);
                cout << "--------- job 1 is ended -----------\n";
            }
            if(ob2.is_used()){
                //jesli obiekt 2 zajmuje drukarke, zwalniamy jej ob2.set_printing(0);
                cout << "--------- job 2 is ended -----------\n";
            }
        }
        //-------------------------------------------------------------
        if(key%3 == 0){
            wolna,
            iteracje
        }
        //Jesli wartosc key dzieli sie przez 3 i drukarka jest
        //zadanie 1 zajmuje drukarke if(!inuse(ob1, ob2))
        {
            ob1.set_printing(1);
            it_used = it; //ustawiamy it_used na biezaca
        }else if(key%3 == 1){
            //Jesli wartosc key dzieli sie przez 3 z resta 1 i drukarka jest wolna,
            //zadanie 2 zajmuje drukarke if(!inuse(ob1, ob2))
            {
                iteracje
            }
        }else{
        }
        ob2.set_printing(1);
        it_used = it; //ustawiamy it_used na biezaca
        if(ob1.is_used())
        cout << "printer is used by job 1\n"; else if(ob2.is_used())
        cout << "printer is used by job 2\n";
        else
        cout << " free\n";
        it++;
    }
    system("pause"); return 0;
```

3. Stworzyć nowy projekt. W projekcie stworzyć klasę A, która zawiera wskaźnik typu char do wiersza tekstowego.
    a. Konstruktor sparametryzowany pobiera z listy argumentów formalnych wiersz tekstowy, oblicza ile elementów zawiera ten wiersz, dynamicznie alokuje pamięć i kopiuje wiersz tekstowy do składowej klasy.
    b. Konstruktor domyślny inicjuje wskaźnik do wiersza tekstowego na NULL.
    c. Konstruktor kopiujący, który przyjmuje, jako argument referencję do obiektu tej samej klasy. Oblicza ile elementów zawiera wiersz obiektu przekazywanego przez referencję, dynamicznie alokuje pamięć i kopiuje obiekt do składowej.
    d. Klasa zawiera metodę *disp()*, która wyświetla wiersz na monitorze.
    e. Każdy z konstruktorów wyświetla na monitorze jeden z komunikatów:
        i. „ konstruktor domyślny”
        ii. „konstruktor sparametryzowany”
        iii. „konstruktor kopiujący”
    b. Analogicznie, destruktor wyświetla komunikat: „destruktor”
    I. Dodaj funkcję *fun()* , która przyjmuje obiekt typu A i wyświetla na monitorze jego wiersz tekstowy.
    II. Dodaj funkcję przeciążoną *fun()*, tak żeby funkcja pobierała wskaźnik do obiektu A i również wyświetlała jego wiersz tekstowy.
    III. Policz ilość wywołań konstruktorów i destruktorów.
    IV. Funkcja *main* ma mieć postać:

```
int _tmain(int argc, _TCHAR* argv[])
{
	A ob("aaaaa");
	fun(ob);
	fun(&ob);
	system("pause");
	return 0;
}
```

    V. Dodaj funkcję o nazwie *fun()*, która zwraca obiekt typu A. Funkcja nie przyjmuje żadnych argumentów. Obiekt typu A ma być zadeklarowany w funkcji, jako obiekt lokalny. Teraz funkcja *main* ma wyglądać tak:

```
int _tmain(int argc, _TCHAR* argv[])
{
	//A ob("aaaaa");
	//fun(ob);
	//fun(&ob);
	A cc = fun();
	system("pause");
	return;
}
```

    VI. Ponownie zlicz ilość wywołań konstruktorów i destruktorów.