Zadanie 1

Definiujemy jedno wyliczenie i trzy C-struktury

```
enum Banks {PKO, BGZ, BRE, BPH};

struct Account {
    Banks   bank;
    int  balance;
};

struct Person {
    char   name[20];
    Account account;
};

struct Couple {
    Person  he;
    Person she;
};
```

W funkcji **main** tworzymy tablicę par (**Couple**) z danymi, na przykład, takimi

No	He			She		
	Name	Bank	Balance	Name	Bank	Balance
0	Johny	PKO	1100	Mary	BGZ	1500
1	Peter	BGZ	1400	Suzy	BRE	1300
2	Kevin	PKO	1600	Katy	BPH	1500
3	Kenny	BPH	1800	Lucy	BRE	1700

Zdefiniować funkcję o nagłówku

```
const Couple* bestClient(const Couple* cpls,
                         int size, Banks bank);
```

która zwraca wskaźnik do tej pary (**Couple**) z tablicy przekazanej jako pierwszy argument (o wymiarze size), która ma największą sumę oszczędności jego (he) i jej (she), ale tylko spośród takich par, w których przynajmniej jedno z małżonków ma konto w banku bank. Jeśli żadna z osób nie ma konta w banku bank, to funkcja zwraca **nullptr**. Nie wolno zakładać, że stan konta jest nieujemny; może być dowolnie duży dodatni i dowolnie duży ujemny.
Na przykład program o schemacie

download *BanksPersons.cpp*

```
int main() {
    using std::cout; using std::endl;
    Couple cpls[] = {
        // ...
    };
    const Couple* p = bestClient(cpls,4,BGZ);
    if (p != nullptr)
        cout << p->he.name << " i " << p->she.name
             << ": " << p->he.account.balance +
                        p->she.account.balance << endl;
    else cout << "Nothing found" << endl;
}
```

powinien wypisać coś w rodzaju

```
Peter and Suzy: 2700
```

Zadanie 2

Definiujemy dwie struktury, jedną opisujące punkty na płaszczyźnie i drugą opisującą prostokąty, reprezentowane przez dwa punkty: dolny-lewy (LL – lower-left) i górny-prawy (UR – upper-right) wierzchołek (rozpatrujemy tylko prostokąty, których boki są równoległe do osi współrzędnych):

```
struct Point {
    int x, y;
};

struct Rect {
    Point LL, UR;
};
```

Napisz i przetestuj funkcje:

```
void printPoint(const Point* p);
void printRect(const Rect& r);
bool doIntersect(const Rect* r1, const Rect* r2);
Rect intersection(const Rect& r1, const Rect& r2);
```

(zauważ, że używane są tu zarówno wskaźniki jak i referencje) o następującej funkcjonalności:

- **printPoint** — wypisuje informacje o punkcie;
- **printRect** — wypisuje informacje o prostokącie; może korzystać z **printPoint**);
- **doIntersect** — sprawdza, czy dwa podane prostokąty mają część wspólną;
- **intersection** — zwraca przez wartość prostokąt będący częścią wspólną (przecięciem) dwóch podanych prostokątów, lub zgłasza wyjątek, jeśli podane dwa prostokąty nie mają części wspólnej, np.
  ```
  throw "Error in 'intersection'";
  ```

Na przykład, następująca funkcja **main**

download *Rectangles.cpp*

```
int main() {
    using std::cout;

    Rect ra{{-1,-2},{ 1, 1}};
    Rect rb{{ 0, 0},{ 5, 1}};
    Rect rc{{ 2,-1},{ 4, 2}};
    Rect rd{{ 1,-3},{ 4, 3}};

    cout << std::boolalpha << "Do intersect:\n";
    cout << "ra rb ? " << doIntersect(&ra,&rb) << '\n';
    cout << "ra rc ? " << doIntersect(&ra,&rc) << '\n';
    cout << "rb rd ? " << doIntersect(&rb,&rd) << '\n';

    cout << "Intersections:\n";
    cout << "ra & rb: ";
    printRect(intersection(ra,rb));
    cout << "rb & rc: ";
    printRect(intersection(rb,rc));
    cout << "rc & rd: ";
    printRect(intersection(rc,rd));
    cout << "ra & rd: ";
    printRect(intersection(ra,rd));
}
```

powinna wydrukować

```
Do intersect:
ra rb ? true
ra rc ? false
rb rd ? true
Intersections:
ra & rb: [(0,0),(1,1)]
rb & rc: [(2,0),(4,1)]
rc & rd: [(2,-1),(4,2)]
ra & rd: [(1,-2),(1,1)]
```

UWAGA:
Algorytm pozwalający znaleźć przecięcie jest bardzo prosty: położenie lewej krawędzi przecięcia to maksimum z położeń lewych krawędzi obu prostokątów, położenie prawej krawędzi przecięcia — minimum z położeń prawych krawędzi. Jeśli lewa wypadnie na prawo od prawej, to przecięcie nie istnieje. Podobnie dla krawędzi górnej i dolnej. Dopuszczamy zdegenerowane prostokąty o zerowej szerokości i/lub wysokości.
