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	1200	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. Jeśli dwie lub więcej par spośród tych, które spełniają narzucony warunek ma takie same, największe, oszczędności, to funkcja zwraca wskaźnik do dowolnej z nich.
Na przykład program o schemacie

download *BanksPersons.cpp*

```
#include <iostream>

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;
};

const Couple* bestClient(const Couple* cpls,
                         int size, Banks bank) {
    // ...
}

int main() {
    using std::cout; using std::endl;
    Couple cpls[] = {
        // ...
    };

    const Couple* p = bestClient(cpls,4,BGZ);
    cout << p->he.name << " and " << p->she.name
         << ": " << p->he.account.balance +
                    p->she.account.balance << endl;
}
```

powinien wypisać coś w rodzaju

```
Peter and Suzy: 2700
```

Zadanie 2

Napisz (i przetestuj) opisany niżej program:

Struktura opisująca węzeł listy ma postać

```
struct Node {
    int   data;
```

```
    Node* next;
};
```

Każdy węzeł przechowuje dane — w tym przypadku po prostu liczbę całkowitą data. Napisz następujące funkcje:

```
1    Node* arrayToList(const int arr[], size_t size);
2    Node* removeOdd(Node* head);
3    void showList(const Node* head);
4    void deleteList(Node*& head);
```

gdzie

1. **arrayToList** pobiera tablicę **int**'ów i jej wymiar. Zadaniem funkcji jest utworzenie listy jednokierunkowej obiektów struktury **Node**, zawierającej w kolejnych węzłach kolejne liczby z przekazanej tablicy (w takiej samej kolejności!). Funkcja zwraca wskaźnik do „głowy" utworzonej listy.
2. **removeOdd** pobiera wskaźnik do „głowy" listy i zwraca wskaźnik do „głowy" listy powstającej z listy pierwotnej po usunięciu wszystkich węzłów, w których data jest liczbą nieparzystą.
   UWAGA: Funkcja ta *nie* powinna tworzyć żadnych nowych węzłów, tylko usuwać te zawierające nieparzyste dane. Jeśli lista zawiera same liczby nieparzyste, wszystkie węzły powinny zostać usunięte, a funkcja powinna zwrócić **nullptr**. Zapewnić, by przy każdym usuwaniu węzła funkcja drukowała wartość danej w nim zawartej, abyśmy widzieli, że rzeczywiście węzły te są usuwane.
3. **showList** drukuje zawartość listy (dane z kolejnych węzłów, w jednej linii, oddzielone znakami odstępu).
4. **deleteList** usuwa wszystkie węzły listy; wskaźnik do „głowy" przesłany jest przez referencję, aby funkcja mogła zmienić jego oryginał (na **nullptr**, co odpowiada liście pustej). Funkcja wypisuje informacje o usuwanych węzłach.

Przykładowy schemat programu:

download *ListyNoTempl.cpp*

```
#include <iostream>

struct Node {
    int   data;
    Node* next;
};

Node* arrayToList(const int arr[], size_t size) {
        // ...
}

Node* removeOdd(Node* head) {
    // ...
}
```

```
void showList(const Node* head) {
    // ...
}

void deleteList(Node*& head) {
    // ...
}

int main() {
    int arr[] = {1,2,3,4,5,6};
    size_t size = sizeof(arr)/sizeof(*arr);
    Node* head = arrayToList(arr,size);
    showList(head);
    head = removeOdd(head);
    showList(head);
    deleteList(head);
    showList(head);
}
```

Program napisany według tego schematu powinien wydrukować:

```
1 2 3 4 5 6
DEL:1 DEL:3 DEL:5
2 4 6
del:2 del:4 del:6
Empty list
```

Zadanie 3

Napisz (i przetestuj) opisany niżej program:

Struktura opisująca węzeł listy ma postać

```
struct Node {
    int   data;
    Node* next;
};
```

(każdy węzeł przechowuje dane — w tym przypadku po prostu liczbę całkowitą data).

1. Napisz funkcję o nagłówku

   ```
   void add(Node*& head, int data);
   ```

   pobierającą wskaźnik do „głowy” listy (nullptr reprezentuje listę pustą) i daną całkowitą a dodającą węzeł listy z daną data tak, żeby węzły na liście pozostawały cały czas uporządkowane według przechowywanych w składowej data wartości (w kolejności rosnącej).
2. Napisz funkcję o nagłówku

   ```
   bool any(const Node* head, function<bool(int)> pred);
   ```

pobierającą wskaźnik do „głowy” listy i zwracającą **true** wtedy i tylko wtedy, gdy dana zawarta w *jakimkolwiek* węźle listy spełnia predykat pred (to znaczy zwraca dla niego **true**).

3. Napisz funkcję o nagłówku

   ```
   bool all(const Node* head, function<bool(int)> pred);
   ```

   pobierającą wskaźnik do „głowy” listy i zwracającą **true** wtedy i tylko wtedy, gdy dana zawarta w *każdym* węźle listy spełnia predykat pred.
4. Napisz funkcję o nagłówku

   ```
   void deleteList(Node*& head);
   ```

   zwalniającą (za pomocą **delete**) wszystkie węzły listy do której wskaźnik przekazany został jako argument. Przy każdym usuwaniu węzła funkcja powinna drukować wartość danej z usuwanego węzła, abyśmy widzieli, że rzeczywiście węzły te są usuwane. Po powrocie z funkcji head powinno być wskaźnikiem pustym (bo reprezentuje listę, która stała się pusta).
5. Napisz funkcję o nagłówku

   ```
   void printList(const Node* head);
   ```

   wypisującą dane z kolejnych węzłów listy w jednym wierszu.

Zauważ, że funkcje **add** i **deleteList** pobierają wskaźnik do „głowy” listy przez referencję, by mogły ten wskaźnik zmodyfikować.
Przykładowy schemat programu:

```
#include <iostream>
#include <functional>
using std::cout; using std::endl;
using std::cerr; using std::function;

struct Node {
    int   data;
    Node* next;
};

void printList(const Node* head) {
    // ...
}

void add(Node*& head, int data) {
    // ...
}

bool any(const Node* head, function<bool(int)> pred) {
    // ...
}
```

```
bool all(const Node* head, function<bool(int)> pred) {
    // ...
}

void deleteList(Node*& head) {
    // ...
}

int main() {
    Node* head = 0;
    add(head,3);
    add(head,6);
    add(head,2);
    add(head,8);
    add(head,5);
    printList(head);
    cout << std::boolalpha;
    cout << "less than 10 all "
        << all(head,[] (int i) -> bool {return i< 10;})
        << endl;
    cout << "is even all      "
        << all(head,[] (int i) -> bool {return i%2 == 0;})
        << endl;
    cout << "is even any      "
        << any(head,[] (int i) -> bool {return i%2 == 0;})
        << endl;
    deleteList(head);
}
```

Program powinien wydrukować coś w rodzaju:

```
2 3 5 6 8
less than 10 all true
is even all      false
is even any      true
Del: 2 Del: 3 Del: 5 Del: 6 Del: 8
```
