GRUPA II (2.XII.25)

1. Funkcja `int max_wyst(wel L)` zwraca jako wartość ten element nieuporządkowanej listy prostej `L`, który występuje na niej największą liczbę razy (jeśli jest więcej takich elementów, funkcja powinna zwracać pierwszy z nich). Np. dla listy $3 \to 2 \to 4 \to 1 \to 2 \to 1 \to 1 \to 3 \to 2$ wynikiem powinno być 2, a dla listy nie zawierającej powtórzeń — jej pierwszy element.

2. Dane są dwie uporządkowane listy proste `P` i `Q` bez wartownika, przy czym pierwsza może zwierać powtórzenia, a druga nie. Zadaniem funkcji `void usun(wel *P, wel Q)` jest usunięcie z `P` wszystkich wystąpień elementów z `Q`. Można posłużyć się wywołaniem funkcji `usun_pierwszy(&L)` z ćwiczeń.

3. Napisz funkcję `void odwróć(wel2 *L)`, króra dokonuje odwrócenia listy dwukierunkowej `L`.

   Uwaga: odwracanie listy dwukierunkowej polega na zamianie wartości pól `q->poprz` i `q->nast` w kolejnych węzłach listy. W rezultacie wskaźnik `*L` powinien odsyłać do dawnego ostatniego elementu listy, który teraz jest pierwszy.

4. Napisz funkcję `void usun_l(weld *T)`, która usuwa wszystkie liście z drzewa `*T`. Jeśli korzeń jest liściem, czyli jedynym elementem drzewa, wynikiem jest drzewo puste.

5. Funkcja `int max_poz(weld T, int k)` ma za zadanie zwrócić wartość maksymalnego elementu wśród węzłów poziomu `k` w *nieuporządkowanym* drzewie `T` (korzeń drzewa ma poziom 1). Zakładamy, że drzewo zawiera wyłącznie liczby dodatnie. Jeśli w wywołaniu `max_poz` paramert `k` ma wartość większą niż wysokość drzewa, funkcja zwraca wartość 0.

GRUPA III (4.XII.25)

1. Napisz kod funkcji `int segment(wel L)` operującej na jednokierunkowej liście nieuporządkowanej `L`, która zwraca długość najdłuższego segmentu w `L`, tj. ciągu jej elementów utworzonego przez kolejne liczby `int`. Np. dla listy

   $$1 \to 2 \to 3 \to 5 \to 3 \to 2 \to 3 \to 4 \to 5 \to 3 \to \texttt{NULL}$$

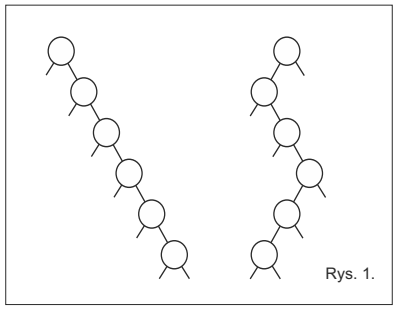
   funkcja powinna zwrócić wartość 4, bo najdłuższy segment to $2 \to 3 \to 4 \to 5$.

2. Funkcja `wel przekroj(wel P, wel Q)` zwraca wskaźnik na początek nowej listy, która zawiera wszystkie elementy należące jednocześnie do `P` i `Q`. Listy `P` i `Q` są nieuporządkowane i nie zawierają powtórzeń. Funkcja `przekroj` pozostawia je bez zmian. Można posłużyć się wywołanim funkcji `dodajp(&L, y)` z ćwiczeń (dodającej nowy element na początku `L`).

3. Napisz funkcję logiczną `int bez_powt(wel L)`, która zwraca wartość 1, gdy jednokierunkowa nieuporządkowana lista cykliczna wskazywana przez `L` nie zawiera powtarzających się elementów, lub 0 w przeciwnym razie.

4. Napisz kod rekursywenej funkcji `int suma_liści(weld T)`, która jako wartość zwraca sumę liczb zapisanych w liściach drzewa `T`. Gdy drzewo jest puste, funkcja zwraca wartość 0.

5. Napisz funkcję `int degen(weld T)`, która zwraca wartość 1 jeśli drzewo `T` ma zdegenerowaną strukturę, efektywnie identyczną ze zwykłą listą jednokierunkową (tak jak na rysunku). Przyjmujemy, że lista pusta jest zdegenerowana.


Rys. 1.

ROZWIĄZANIA — GRUPA II

1.
```
int max_wyst(wel L){
   int k=0, l, mx;
   int p,q;
   for (p=L; p!=NULL; p=p->nast){
      l = 1;
      for (q=p->nast; q!=NULL; q=q->nast) if (q->x == p->x) l++;
      if (k<l){
         k = l;
         mx = p->x;
      }
   }
   return mx;
}
```

2.
```
wel usun(wel *P, wel Q){
   while (Q!=NULL){
      while ((*P!=NULL) && ((*P)->x < Q->x)) P = &((*P)->nast);
      while ((*P!=NULL) && ((*P)->x == Q->x)) usun_pierwszy(P);
      Q = Q->nast;
   }
}

void usun_pierwszy(wel *P){
   wel q=*P;
   if (q!=NULL){
      *P = q->nast;
      free(q);
   }
}
```

3. Rozwiązanie iteracyjne

```
void odwróć(wel2 *L){
   wel2 p=*L;
   while (p != NULL) {
      *L = p;
      p = p->nast;
      (*L)->nast = (*L)->poprz;
      (*L)->poprz = p;
   }
}
```

Rozwiązanie rekurencyjne

```
wel2 odwróć_r(wel2 L){
   wel2 p=L;
   if (L != NULL) {
       if (L->nast != NULL) {
           p = odwroc2r(L->nast);
           L->nast->nast = L;
       }
       L->poprz = L->nast;
       L->nast = NULL;
   }
   return p;
}
```

4.
```
void usun_l(weld *T){
   if (*T!=NULL) {
      if (((*T)->l == NULL) && ((*T)->p == NULL)) { free(*T); *T=NULL; }
      else {
         usun_l(&((*T)->l));
         usun_l(&((*T)->p));
      }
   }
}
```

5.
```
int max_poz(weld T, int k){
   if ((T==NULL) || (k<1)) return 0;
   return (k==1) ? T->x : max( max_poz(T->l, k-1), max_poz(T->p, k-1) );
}
```

ROZWIĄZANIA — GRUPA III

1.
```
int segment (wel L){
   int s=0, ms=0, y;
   if (L!=NULL){
      y = L->x;
      L = L->nast;
      s=1;
      while (L!=NULL){
         if (L->x==y+1){ s++;}
         else {
            if (s>ms) ms=s;
            s=1;
         }
         y=L->x;
         L=L->nast;
      }
   }
   return (s>ms) ? s : ms;
}
```

2.
```
wel przekroj(wel P, wel Q){
   wel K=NULL;
   while (P!=NULL){
       if ( znajdz(P->x, Q) ) dodajp(&K, P->x);
       P = P->nast;
   }
   return K;
}

int znajdz(int z, wel S){
   while (S!=NULL) {
      if (S->x == z) return 1;
      S = S->nast;
   }
   return 0;
}
```

3.
```
int bez_powt(wel L){
   wel p, q;
   if (L != NULL){
      p = L;
      do {
         for (q=p->nast; q!=L; q=q->nast)
            if (q->x == p->x) return 0;
         p = p->nast;
      } while (p != L)
   }
   return 1;
}
```

4.
```
int suma_liści(weld T){
    if (T==NULL) return 0;
    if ((T->l==NULL) && (T->p==NULL)) return T->x;
    return suma_liści(T->l) + suma_liści(T->p);
}
```

5.
```
int degen(weld T){
    if (T==NULL) return 1;
    return ((T->l==NULL) && degen(T->p)) || ((T->p==NULL) && degen(T->l));
```