Wstęp do programowania

Wykład 5. Struktury

Plan wykładu

1. Struktury.

- przykłady zastosowania.

2. Zarządzanie pamięcią.

3. Podstawowe struktury danych

- listy,

- listy dwukierunkowe.

Struktury

```
struct Complex{
  double Re;
  double Im;
};


void main(void){
  struct Complex z1, z2;
  z1.Re = 7;
  z1.Im = 12;
  z2.Re = 28;
  z2.Im = 4;
}
```

Struktury umożliwiają konstrukcję złożonych typów danych na bazie istniejących typów.

Struktury

```
struct Compex add1(struct Complex z1, struct Complex z2){
  struct Complex res;
  res.Re = z1.Re + z2.Re;
  res.Im = z1.Im + z2.Im;
  return res;
}
```

A używając wskaźników (wydajniej).

```
void add2(struct Complex *z1, struct Complex *z2, struct Complex *res){
  res->Re = z1->Re + z2->Re;
  res->Im = z1->Im + z2->Im;
}
```

Wyrażenia `a.b` i `(&a)->b` są równoważne.

Struktury

A jeszcze inaczej:

```
struct Complex *add3(struct Complex *z1, struct Complex *z2){
  struct Complex *res;
  res = (struct Complex *)malloc(sizeof(struct Complex));
  res->Re = z1->Re + z2->Re;
  res->Im = z1->Im + z2->Im;
  return res;
}
```

Funkcja `malloc()` rezerwuje pamięć na obiekt o rozmiarze `struct Complex`. Na zakończenie procedura zwraca wskaźnik do tej zarezerwowanej pamięci zawierającej wynik dodawania. Pamięć należy później (gdy już wynik nie będzie potrzebny) zwolnić funkcją `free()`.

Struktury – przykład: równanie kwadratowe

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

struct Results{
  int number;
  double x1;
  double x2;
};
```

Struktury – przykład: równanie kwadratowe

```
struct Results findRoots(double a, double b, double c){
  struct Results result;
  double delta = b*b-4*a*c;
  if(delta<0.0){
    result.number = 0;
  }else if (delta==0.0){
    result.number = 1;
    result.x1 = result.x2 = b/(2*a);
  }else{
    result.number = 2;
    result.x1 = (-b+sqrt(delta))/(2*a);
    result.x2 = (-b-sqrt(delta))/(2*a);
  }
  return result;
}
```

Struktury – przykład: równanie kwadratowe

```
void main(int argc, char**argv){
  if(argc!=4)  return;
  double a = atof(argv[1]);
  if (a==0.0)  return;
  double b = atof(argv[2]);
  double c = atof(argv[3]);
  struct Results res = findRoots(a, b, c);
  printf("liczba rozwiazan rownania %lf*x*x + %lf*x + %lf = 0
          wynosi %d\n", a, b, c, res.number);
  switch (res.number){
    case 2:
      printf("x2 = %lf\n", res.x2);
    case 1:
      printf("x1 = %lf\n", res.x1);
  }
}
```

Struktury: dynamiczna tablica

```
#include<stdio.h>
#include<stdlib.h>
void main(void){
  double tablica[10]; // musimy podać rozmiar tablicy
  int i = 0;
  double a;
  do{
    scanf("%lf", &a);
    tablica[i++] = a; // co będzie, gdy liczby się nie zmieszczą?
  }while(a>=0);
  for(int i=0; i<10; i++){
    printf("%lf\n", tablica[i]);
    if(tablica[i]<0)
      break;
  }
}
```

Struktury: dynamiczna tablica

Spróbujemy zrobić „dynamiczną” tablicę, której rozmiar będzie się powiększał w miarę dodawania do niej kolejnych elementów.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

struct DynamicArray{
  int size;  // rozmiar tablicy values
  int items; // liczba przechowywanych elementow
  double *values; // wskaznik do tablicy elementow
};
```

Dynamiczna tablica - alokacja pamięci

```
void add(struct DynamicArray *array, double v){
  if (array->items == array->size){
    array->size = (int)(2.0*array->size)+1;
    double newValues[array->size];
    for(int i=0; i<array->items; i++)
      newValues[i] = array->values[i];
    array->values = newValues;
  }
  array->values[(array->items)++] = v;

}
```

To rozwiązanie jest **niepoprawne**, gdyż `newValue` jest zmienną lokalną i po zakończeniu funkcji `add` ona zniknie. I wtedy `array->values` będzie wskazywało na nieistniejącą tablicę! Trzeba to zrobić inaczej.

Dynamiczna tablica - alokacja pamięci

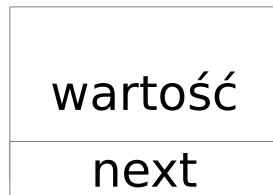
```
double add(struct DynamicArray *array, double v){
  if (array->items == array->size){
    array->size = (int)(2.0*array->size)+1;
    double *newValues = (double *)calloc(array->size,
                                         sizeof(double));
    memcpy(newValues, array->values, array->items*sizeof(double));
    free(array->values);
    array->values = newValues;
    printf("new size %d\n", array->size); // do celow testowych
  }
  array->values[(array->items)++] = v;

}

double get(struct DynamicArray *array, int i){
  return array->values[i];
}
```

Struktury – dynamiczna tablica

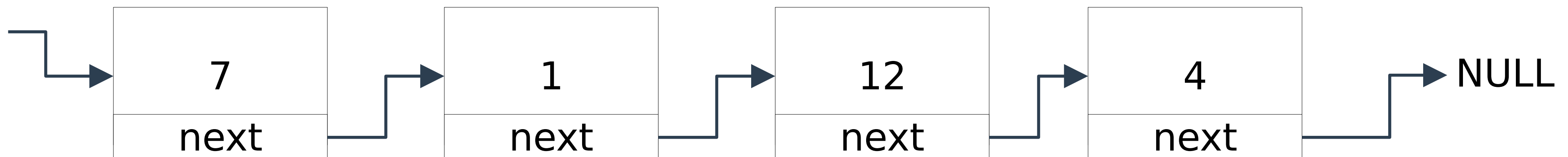
```
void main(void){
  struct DynamicArray tablica = {0, 0, NULL};
  double a;

  do{
    scanf("%lf", &a);
    add(&tablica, a);
  }while(a>=0);

  for(int i=0; i<tablica.items; i++){
    printf("%lf\n", get(&tablica, i));
    if(get(&tablica, i)<0)
      break;
  }
}
```

Lista

Lista to jedna z podstawowych struktur danych używanych w algorytmach
Zwykle jest implementowana za pomocą dwuelementowej struktury.

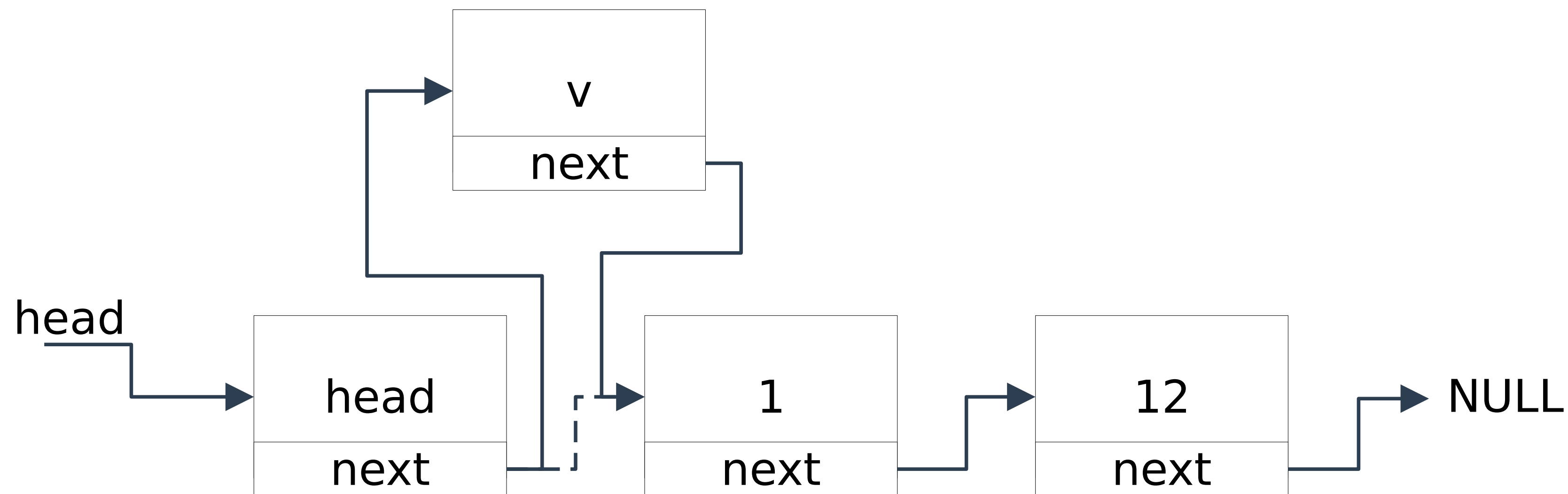



```
struct List{
  double value;
  struct List *next;
};
```



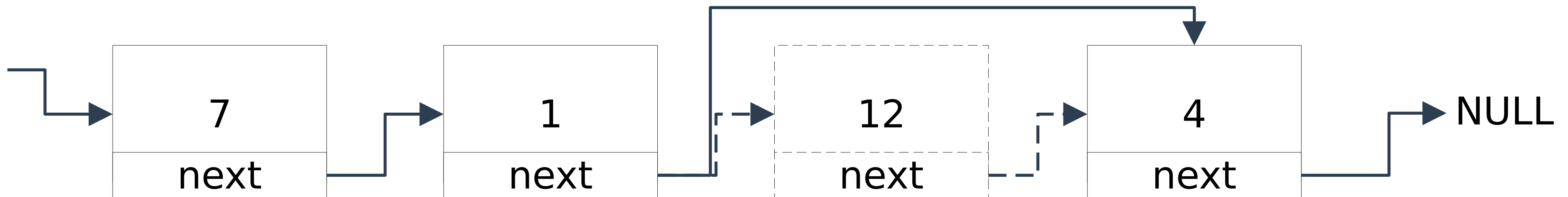
Lista – dodawanie elementu

```
void add(struct List *head, double v){
  struct List *newElement;
  newElement = (struct List *)malloc(sizeof(struct List));
  newElement->value = v;
  newElement->next = head->next;
  head->next = newElement;
}
```



Lista – usuwanie elementu

```
void remove(struct List *prev){
  struct List *removedElement = prev->next;
  prev->next = removedElement->next;
  free(removedElement);
}
```



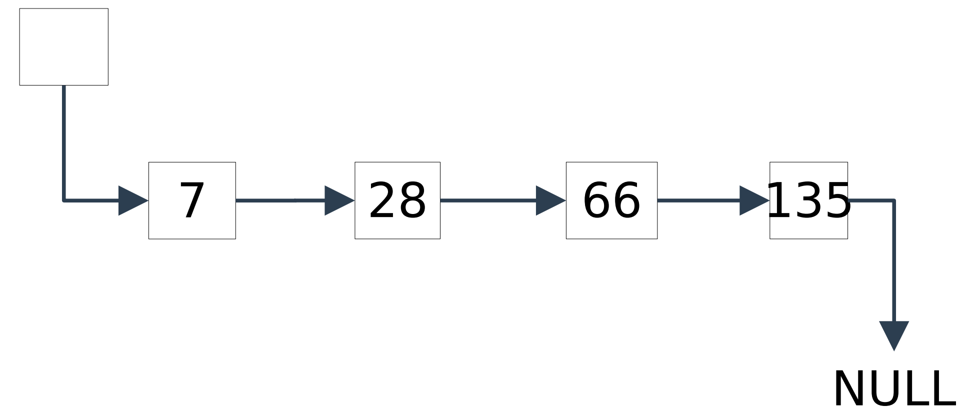
Problem: musimy znać wskaźnik do elementu poprzedzającego. Dlatego tzw. listy jednokierunkowe mają zwykle „głowę” – specjalny, pusty element na początku listy.

Ćwiczenia: napisać funkcje wypisujące zawartość listy oraz zwracające wskaźnik do elementu listy o zadanej wartości

Lista – implementacja kursorowa

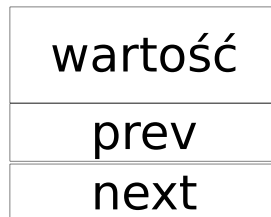
Implementacja kursorowa opiera się na dwuwymiarowych tablicach i była jedyną możliwością implementacji list w językach nie posiadających wskaźników.

indeks	wartość	następny
0	głowa	5
1	66	7
2		
3	28	1
4		
5	7	3
6		
7	135	-1
8		

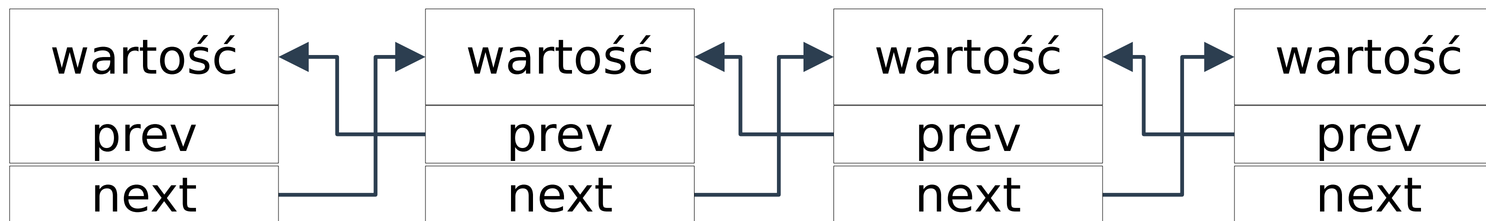


Lista dwukierunkowa

Elementy listy dwukierunkowej posiadają wskaźniki do elementu następnego i poprzedzającego. Dzięki temu w takich listach nie potrzeba głowy i można łatwo usunąć dowolny element.




```
struct List{
  double value;
  struct List *next;
  struct List *prev;
};
```



Zadania: zaimplementować podstawowe operacje na listach.

Dziękuję za uwagę