Struktury łączone

Struktura ze wskaźnikiem na strukturę tego samego typu:

```
typedef struct node{
int value;
struct node* next_node;
} Node;
```

Struktury łączone — lista jednokierunkowa

```
typedef struct node{
  int value;
  struct node* next_node;
} Node;

Node* new_node(int v){
  Node* n = (Node*)malloc(sizeof(Node));
  n->value = v;
  n->next_node = NULL;
  return n;
}

typedef struct list
{
  Node* start_node;
  Node* end_node;
} List;
```

Struktury łączone — lista jednokierunkowa

```
void add(List* l, int x) {
  if(l->start_node == NULL) {
    l->start_node = new_node(x);
    l->end_node = l->start_node;
    return;
  } else {
    Node* t = new_node(x);
    l->end_node->next_node = t;
    l->end_node = t;
  }
}
```

Struktury łączone — lista jednokierunkowa

```
void search(Node* first, int x) {
  while(first->next_node != NULL) {
    if(first->value == x) {
      printf("Znaleziono\n");
      return;
    }
    first = first->next_node;
  }
  if(first->value == x) {
    printf("Znaleziono\n");
    return;
  }
  printf("Nie naleziono\n");
}
```

Cały kod w pliku *lista.c*

Struktury łączone

Za pomocą stuktur łączonych możemy zarezerwować dokładnie tyle pamięci ile potrzeba.

Struktury łączone

Za pomocą stuktur łączonych możemy zarezerwować dokładnie tyle pamięci ile potrzeba.

Lista jednokierunkowa pozwala na te same operacje co tablica, dodatkowo możemy dodawać dynamicznie elementy, ale czas dostępu jest o wiele gorszy: szukanie zajmuje tyle ile długość listy.

Struktury łączone — kolejka

W liście mamy łatwy dostęp do pierwszego/ostatniego elementu.

Struktury łączone — kolejka

W liście mamy łatwy dostęp do pierwszego/ostatniego elementu.

Możemy zrealizować kolejkę FIFO — dodanie el. to wstawienie na koniec, usunięcie to usunięcie z początku.

Struktury łączone — kolejka

W liście mamy łatwy dostęp do pierwszego/ostatniego elementu.

Możemy zrealizować kolejkę FIFO — dodanie el. to wstawienie na koniec, usunięcie to usunięcie z początku.

Przykład w pliku *FIFO.c*.

Inne przykłady

Drzewo binarne:

```
struct TreeNode {
 int val;
 struct TreeNode* left;
 struct TreeNode* right;
};
```

Inne przykłady

Drzewo binarne:

```
struct TreeNode {
 int val;
 struct TreeNode* left;
 struct TreeNode* right;
};
```

Drzewo:

```
struct TreeNode {
  int id;
  struct ListNode* ChildrenList; //poczatek listy ze
                                 //wskaznikami na wierzcholki
};
```

Wskaźnik void*

Możemy zadeklarować wskaźnik bez typu.

```
int x;
void* ptr = &x;
float y;
ptr = &y;

float* ptr_float = ptr;
printf("%f\n", *ptr_float);
printf("%f\n", *(float*)ptr);
```

Wskaźnik void*

Możemy zadeklarować wskaźnik bez typu.

```
int x;
void* ptr = &x;
float y;
ptr = &y;

float* ptr_float = ptr;
printf("%f\n", *ptr_float);
printf("%f\n", *(float*)ptr);
```

Wskaźniki każdego typu w C mają ten sam rozmiar, więc można je konwertować między sobą.
Operacja dereferencji (*) jak i operacje arytmetyczne (inkrementacja, dodawanie) są niezdefiniowane dla wsk. typu `void*`

void* — użycie

Możemy trzymać wskaźnik typu `void*` na dane.
Oszczędność miejsca - jeśli chcemy trzymać w liście tablice, struktury, stringi itp. to używamy tylko tyle miejsca ile jest wymagane.

```
typedef struct node{
  void* data;
  struct node* next_node;
} Node;
```