Informatyka 2

Wprowadzenie do dynamiczny struktur danych
Lista

Paweł Strączyński
pstraczynski@tu.kielce.pl

Dynamiczne struktury danych

Strukturą danych nazywamy „pojemnik na dane”, który układa dane w odpowiedni sposób np. tablica, rekord (struktura).

Dynamiczną strukturą danych nazywamy taką strukturę danych której wielkość i stopień złożoności może zmieniać się podczas działania programu. Podstawowe dynamiczne struktury danych to:

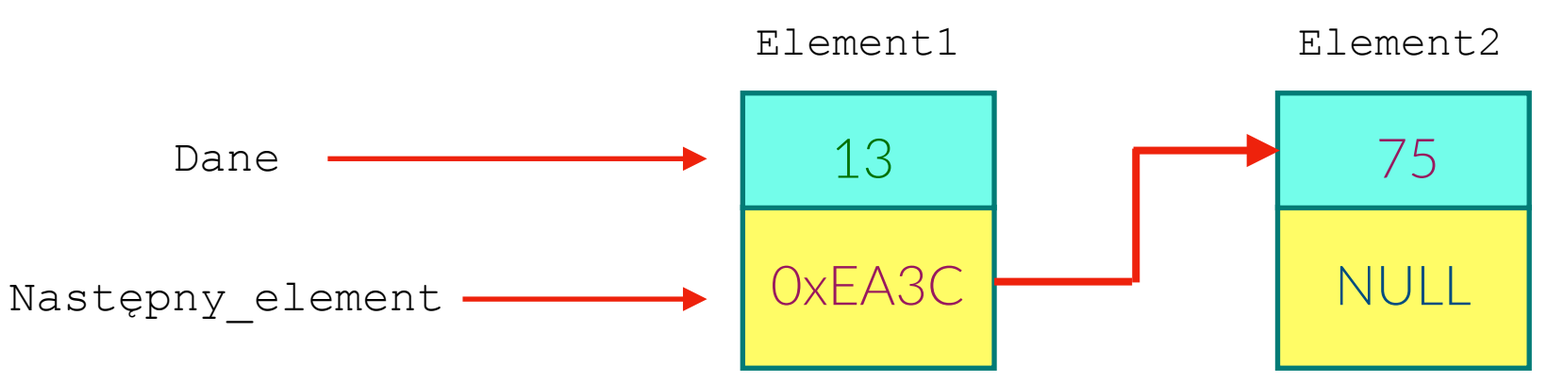
- stos,
- kolejka,
- **lista**,
- drzewo binarne.

Lista

Lista jest strukturą danych, która składa się z ciągu elementów tego samego typu.
Wyróżniamy dwa rodzaje list:

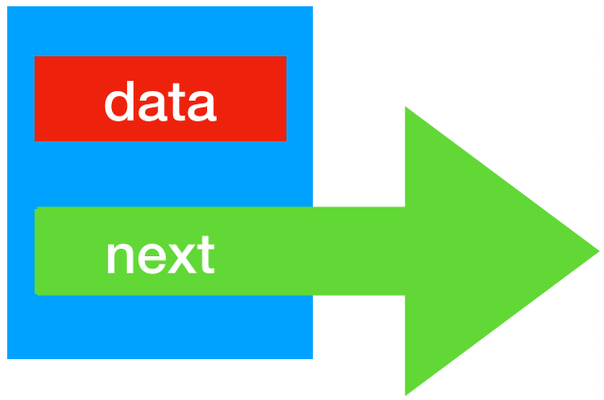
- jednokierunkową - każdy element listy zawiera wskaźnik na następny element - możliwość poruszania się po liście w jedną stronę,
- dwukierunkową - każdy element listy zawiera wskaźnik na następny oraz poprzedni element - możliwość poruszania się po liście w dwie strony,

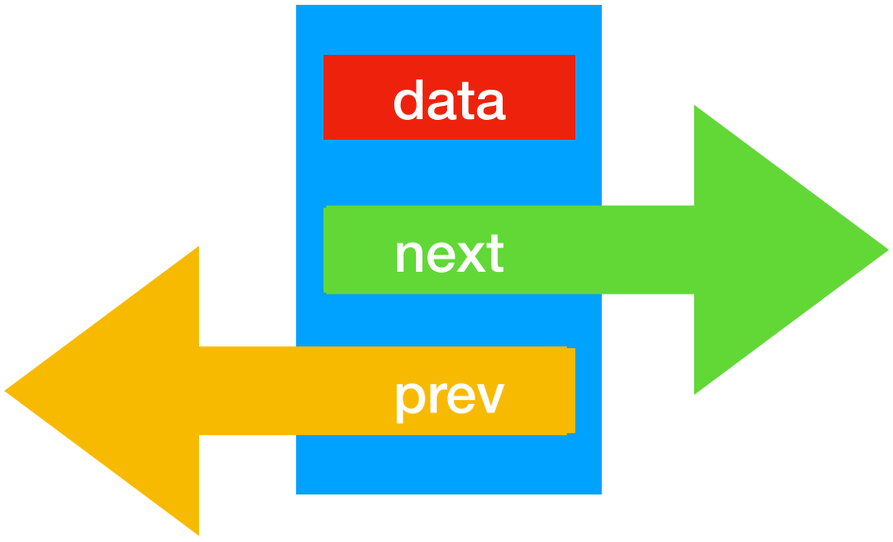
Dostęp do elementów listy jest sekwencyjny – tzn. z danego elementu listy możemy przejść do elementu następnego lub do poprzedniego (w przypadku listy dwukierunkowej).
Dojście do elementu i-tego wymaga przejścia przez kolejne elementy od pierwszego do docelowego.
Elementy listy nie muszą leżeć w pamięci komputera obok siebie. Wskaźnik danego elementu listy wskazuje na kolejny jej element.



Przykład listy jednokierunkowej
Element 1 zawiera wskaźnik na kolejny element listy
Element 2 jest elementem ostatnim listy więc jako następny element wskazuje wartość NULL

Lista jedno- i dwukierunkowa w języku C

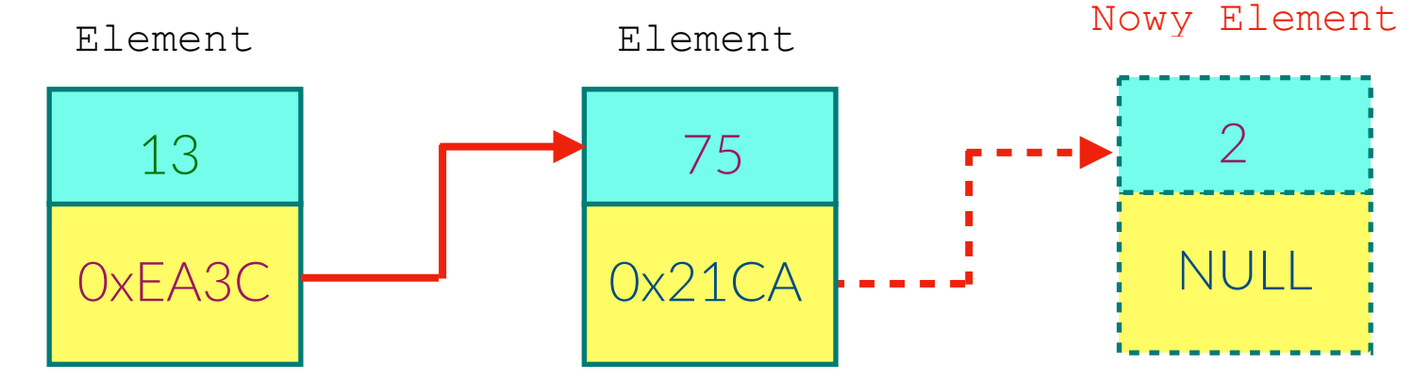


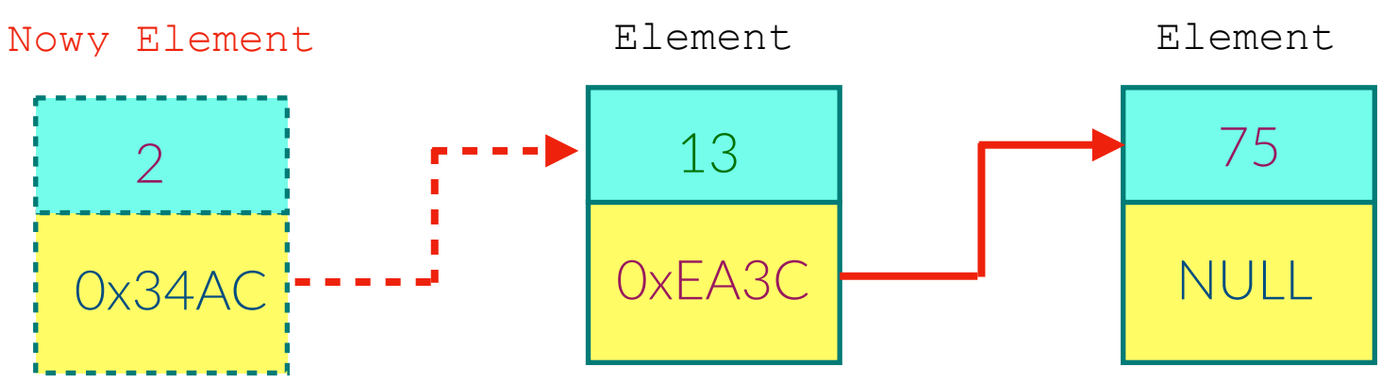



```c
/* lista_jednokierunkowa i dwukierunkowa */


typedef struct Samochod {
   char marka[20];
   char model[20];
   int rok_prod;
} Dane;

typedef struct Element1 { //Lista jednokierunkowa
  Dane dane;
  Element1 *next;
} Element1;

typedef struct Element2 {S //Lista dwukierunkowa
  Dane dane;
  Element2 *next;
  Element2 *prev;
} Element2;

Element1 *lista1 = NULL;
Element2 *lista2 = NULL;
```

Dodanie nowego elementu do listy

Zaletą list jest to, że nowe elementy można szybko dołączać dowolnym miejscu listy. Trzeba jednak tu wyróżnić szczególne przypadki takie jak wstawianie elementu na początku, na końcu oraz za wskazanym już istniejącym elementem listy.





Dodanie nowego elementu na początku listy

Funkcja posiada dwa argumenty:
- wskaźnik na pierwszy element listy
- dane do wstawienia w nowym elemencie

```
/* Dodaj element na początek listy */

int Dodaj_poczatek (Element1 **head, Dane dane)
{
    Element1 *tmp;
    tmp = (Element1*)malloc(sizeof(Element1));
    tmp->dane = dane;
    tmp->next = *head;
    *head = tmp;

    return 0;
}
```

1

Dodanie nowego elementu na początku listy

```
/* Dodaj element na początek listy */
int Dodaj_poczatek (Element1 **head, Dane dane)
{
    Element1 *tmp;
    tmp = (Element1*)malloc(sizeof(Element1));
    tmp->dane = dane;
    tmp->next = *head;
    *head = tmp;

    return 0;
}
```

1

2

Funkcja posiada dwa argumenty:

- wskaźnik na pierwszy element listy
- dane do wstawienia w nowym elemencie

Zmienna pomocnicza w której zostanie zapamiętany adres nowo utworzonej zmiennej dynamicznej

Dodanie nowego elementu na początku listy

```
/* Dodaj element na początek listy */
int Dodaj_poczatek (Element1 **head, Dane dane)
{
    Element1 *tmp;
    tmp = (Element1*)malloc(sizeof(Element1));
    tmp->dane = dane;
    tmp->next = *head;
    *head = tmp;

    return 0;
}
```

1

Funkcja posiada dwa argumenty:

- wskaźnik na pierwszy element listy
- dane do wstawienia w nowym elemencie

2

Zmienna pomocnicza w której zostanie zapamiętany adres nowo utworzonej zmiennej dynamicznej

3

Dane zostają przepisane do pola dane nowo utworzonego elementu listy

Dodanie nowego elementu na początku listy

```
/* Dodaj element na początek listy */

int Dodaj_poczatek (Element1 **head, Dane dane)
{
    Element1 *tmp;
    tmp = (Element1*)malloc(sizeof(Element1));
    tmp->dane = dane;
    tmp->next = *head;
    *head = tmp;

    return 0;
}
```

1

Funkcja posiada dwa argumenty:

- wskaźnik na pierwszy element listy
- dane do wstawienia w nowym elemencie

2

Zmienna pomocnicza w której zostanie zapamiętany adres nowo utworzonej zmiennej dynamicznej

3

Dane zostają przepisane do pola dane nowo utworzonego elementu listy

4

Podmienione zostają dowiązania

- pole `next` nowego pierwszego elementu wskazuje na dotychczasowy pierwszy element,
- wskaźnik na pierwszy element listy zostaje przestawiony na ten nowo dodany

Dodanie nowego elementu za wskazanym elementem

```
/* Dodaj element za wskazanym elementem listy */

int Dodaj_za_elementem (Element1 **current, Dane dane)
{
    Element1 *tmp;
    tmp = (Element1*)malloc(sizeof(Element1));
    tmp->dane = dane;
    tmp->next = (*current)->next;
    (*current)->next = tmp;

    return 0;
}
```

1

2

3

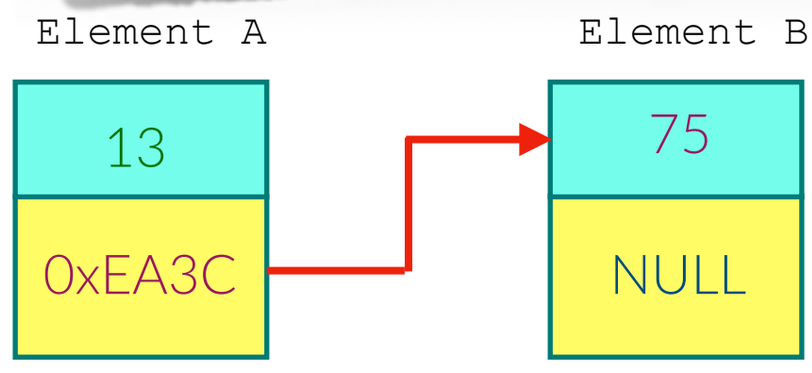
4

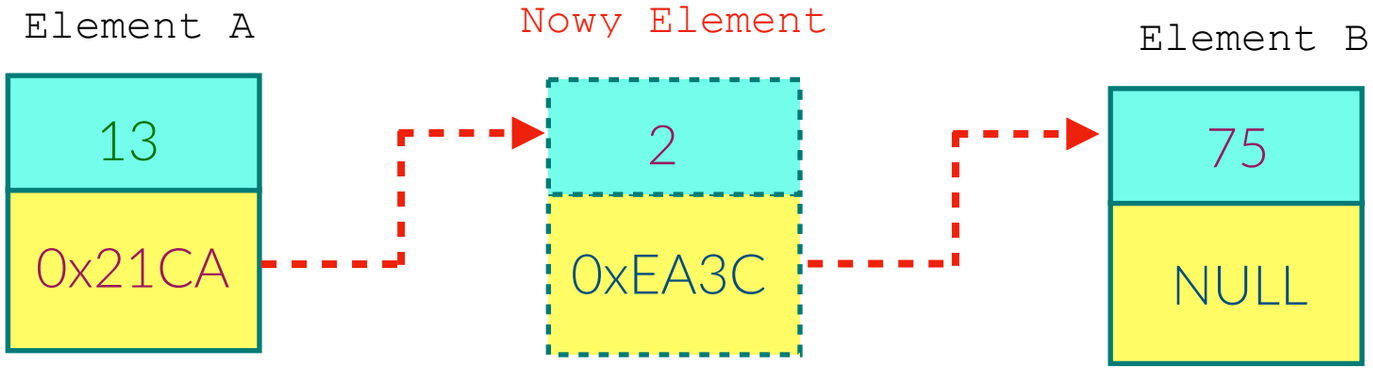
Funkcja posiada dwa argumenty:
- wskaźnik na aktualny element listy
- dane do wstawienia w nowym elemencie

Zmienna pomocnicza w której zostanie zapamiętany adres nowo utworzonej zmiennej dynamicznej

Dane zostają przepisane do pola dane nowo utworzonego elementu listy

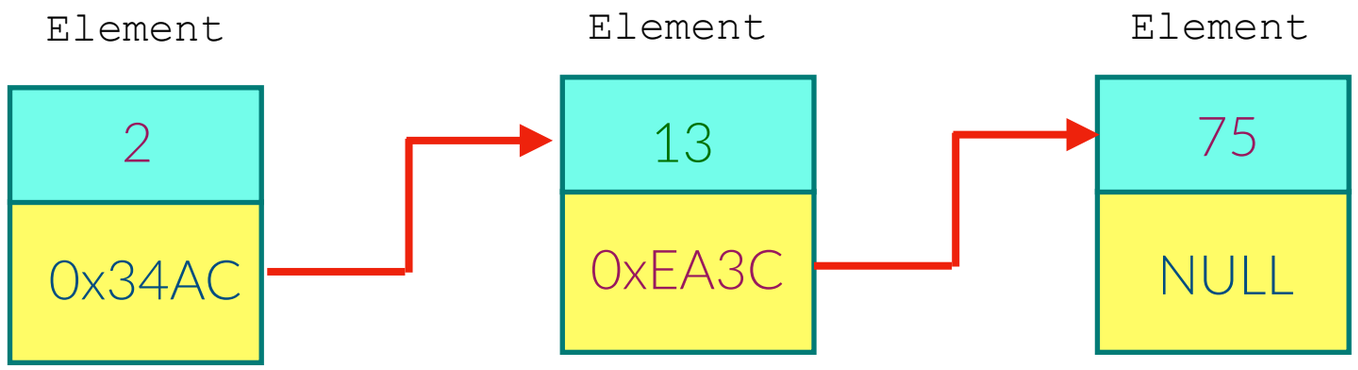
Podmienione zostają dowiązania
- pole `next` nowego elementu wskazuje element który do tej pory był nastepny po tym za którym wstawialiśmy nowy element - Element B,
- wskaźnik na nastepny element w elemencie za którym dokonujemy wstawienia (Element A) zostaje przedstawiony na ten nowo dodany





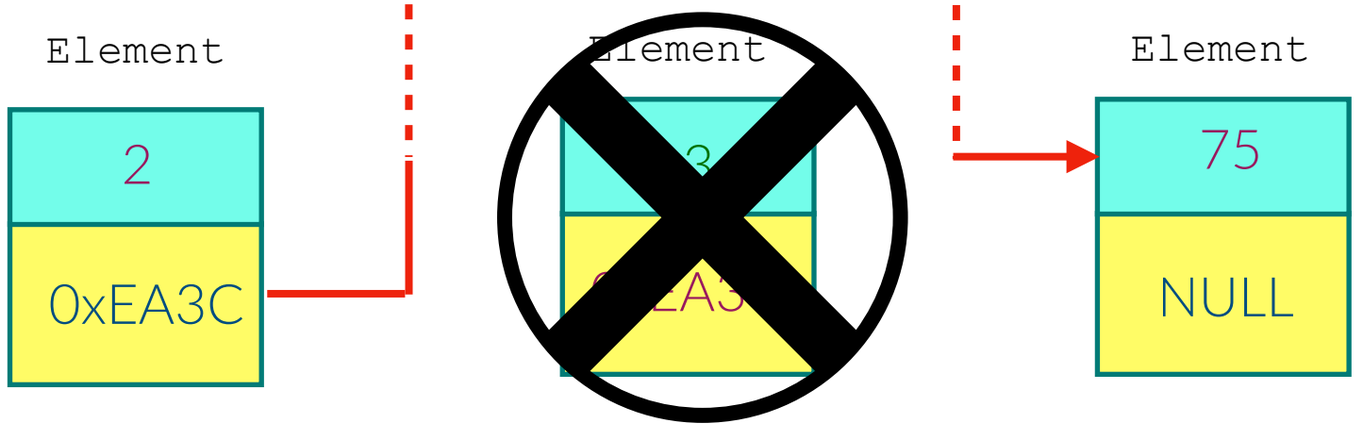
Usuwanie elementu z listy

Kolejną podstawową operacją na liście jest usuwanie elementu. Podobnie jak w przypadku dodawania rozróżnić należy usuwanie pierwszego jak i środkowego lub ostatniego elementu.



Usuwanie elementu z listy

Kolejną podstawową operacją na liście jest usuwanie elementu. Podobnie jak w przypadku dodawania rozróżnić należy usuwanie pierwszego jak i środkowego lub ostatniego elementu.



Usuwanie pierwszego elementu listy

```
/* Usuń pierwszy element listy */

int Usun_poczatek (Element1 **head)
{
    Element1 *tmp;
    tmp = *head;
    *head = tmp->next;
    free(tmp);

    return 0;
}
```

1

2

3

4

Argumentem funkcji jest pierwszy (usuwany) element listy - wskaźnik na listę

Zmienna pomocnicza w której zostanie zapamiętany adres pierwszego elementu listy

Jako pierwszy element listy zostaje ustawiony kolejny (drugi) element listy

Usunięcie zmiennej dynamicznej przechowującej usuwany element listy

Usuwanie elementu za wskazanym elementem

```
/* Usuń kolejny element listy */

int Usun_za_elementem (Element1 **current)
{
    Element1 *tmp;
    tmp = (*current)->next;
    (*current)->next = tmp->next;
    free(tmp);

    return 0;
}
```

1

2

3

4

Argumentem funkcji jest element listy poprzedzający ten usuwany

Zmienna pomocnicza w której zostanie zapamiętany adres elementu poprzedzającego usuwany

Przestawienie wskaźnika z pola `next` poprzedniego elementu przed usuwanym na ten za elementem usuwanym

Usunięcie zmiennej dynamicznej przechowującej usuwany element listy

Drukowanie wszystkich elementów listy

Często na każdym z elementów listy chcemy wykonać jakąś konkretną operację - np. przypisać wartość, porównać zawartość itd. Aby tego dokonać należy przejść po każdym elemencie z listy i wykonać te operację. Poniżej przykład drukowania elementów listy.

```
/* Drukowanie elementów listy */

int Drukuj_liste (Element1 *head)
{
    Element1 *tmp;
    printf(„\nLISTA\n”);
    for( tmp =head; tmp!=NULL; tmp=tmp->next;
    {
        printf(„\n Marka: %s\n”, tmp_>dane.marka);
        printf(„\n Model: %s\n”, tmp_>dane.model);
        printf(„\n Rok produkcji: %s\n”, tmp_>dane.rok_prod);
    }

    return 0;
}
```

1

Argumentem funkcji jest pierwszy element listy

2

Zmienna pomocnicza w której zostanie zapamiętany aktualnie przetwarzany element listy

3

Pętla przechodząca po kolejnych elementach listy od pierwszego aż do ostatniego

4

Wykonanie operacji na elemencie listy - w tym przypadku drukowanie zawartości