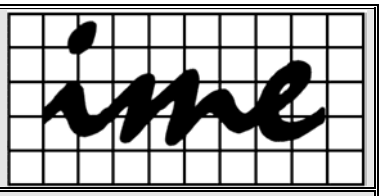
INFORMATYKA

Podstawy programowania w języku C

(Wykład)

INFORMATYKA	
Temat: **Struktury dynamiczne**	Wykład 7

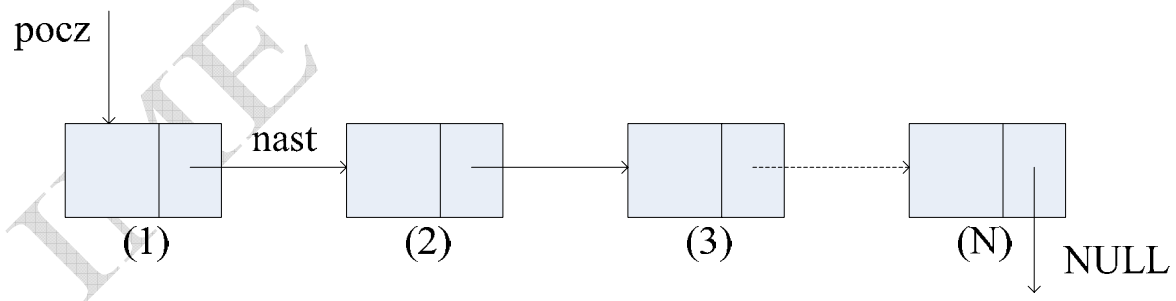
Struktury dynamiczne

- lista jednokierunkowa,
- lista dwukierunkowa,
- listy cykliczne,
- stos,
- kolejka,
- drzewa.

W większości złożonych programów wykorzystuje się pewne struktury przechowywane na tzw. *stercie* (obszar pamięci zajmowany przez kod wynikowy i zmienne globalne a stos). Struktury te nazywane są strukturami dynamicznymi, ponieważ ich wielkość i stopień złożoności zmienia się w czasie działania programu. Do struktur dynamicznych zalicza się *listy*, *stosy*, *kolejki* i *drzewa*.

1. Lista jednokierunkowa

Lista jednokierunkowa zawiera pewne dane (elementy listy) oraz *wskaźnik* na następny element listy. Ostatni element listy jednokierunkowej ma wskaźnik ustawiony na NULL. Dostęp do listy umożliwia wskaźnik ustawiony na jej pierwszy element (na *Rys. 1* jest to wskaźnik „*pocz*"). Przeglądania listy i pobieranie jej elementów możliwe jest dzięki operatorowi wskaźnikowemu „->".



Rys. 1. N-elementowa lista jednokierunkowa

Elementy listy jednokierunkowej są *strukturami*. W *języku ANSI C/C++* jeden element listy jednokierunkowej ma następującą postać:

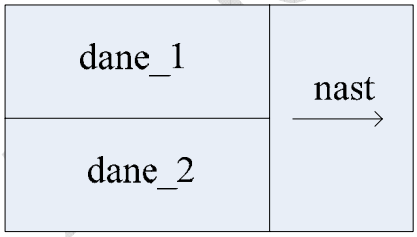
```
struct Lista_jednokierunkowa
{
 int dana_1;
 ...
 int dana_n;
 struct Lista_jednokierunkowa *nast;
};
```

Przykład 87.

W przykładzie następuje wygenerowanie listy jednokierunkowej, ze wstawianiem elementów na początku listy.

1) Następuje zdefiniowanie struktury o dwóch polach „dane_1" i „dane_2" oraz zmiennej wskaźnikowej „nast" wskazującej na następny element listy.

```
struct Element_listy_jednokierunkowej
{
  int dana_1;
  int dana_2;
  struct Element_listy_jednokierunkowej *nast;
};
```



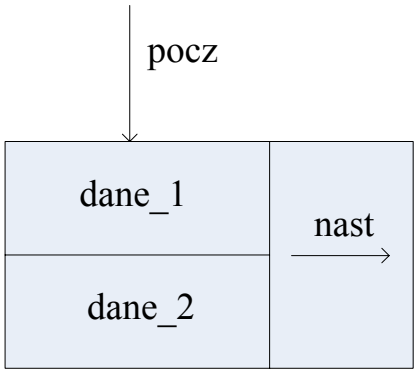
Rys. 2. Budowa elementu listy jednokierunkowej

2) Następuje deklaracja dwóch zmiennych wskaźnikowych „pocz" i „wsk" pomocnych przy operowaniu na elementach listy.

```
struct Element_listy_jednokierunkowej *pocz=NULL, *wsk=NULL;
```

3) Następuje dynamiczne przydzielenie pamięci dla zmiennej wskaźnikowej „pocz". Tym samym zostaje utworzy pierwszy element listy jednokierunkowej o polach zdefiniowanej struktury. Wskaźnik „pocz" zostaje ustawiony na tym elemencie listy.

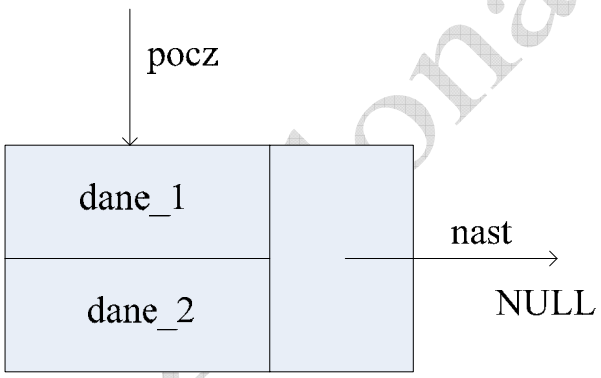
```
pocz=malloc(sizeof(struct Element_listy_jednokierunkowej));
```



Rys. 3. Deklaracja elementu struktury

3) Zgodnie z definicją listy jednokierunkowej następuje przypisanie wartości „NULL" wskaźnikowi „nast" pola struktury.

```
pocz -> nast = NULL;
```

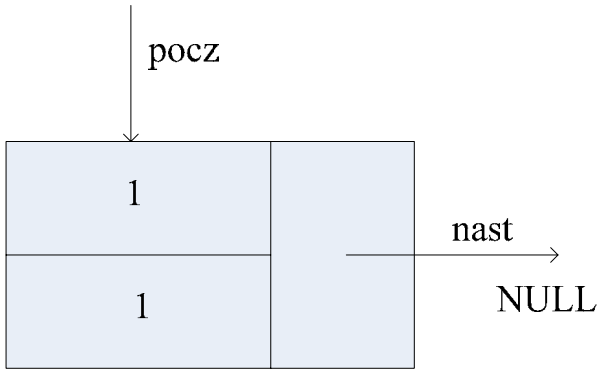


Rys. 4. Tworzenie listy jednokierunkowej

4) Następuje zadanie wartości liczbowych polom „dane_1" i „dane_2".

Np.:

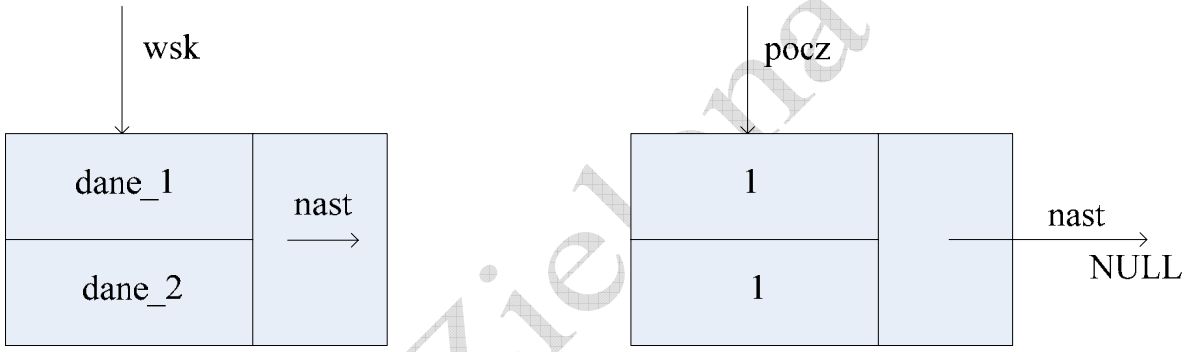
```
pocz->dana_1=1;
pocz->dana_2=1;
```



Rys. 5. Tworzenie listy jednokierunkowej

5) Następuje utworzenie kolejnego elementu listy. Wskaźnik „wsk” pokazuje na nowo utworzony element listy.

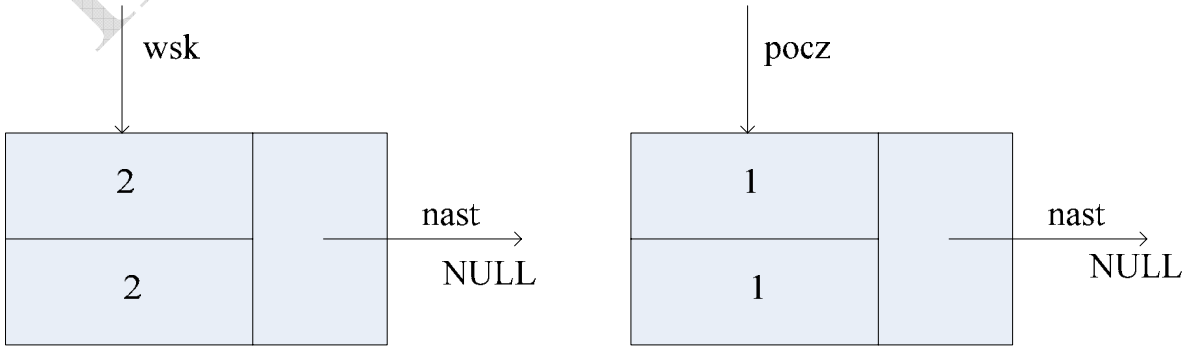
```
wsk=malloc(sizeof(struct Element_listy_jednokierunkowej));
```



Rys. 6. Tworzenie listy jednokierunkowej

6) Następuje przypisanie wartości „NULL” wskaźnikowi „wsk” oraz zadanie wartości liczbowych polom „dane_1” i „dane_2” nowego elementu listy.

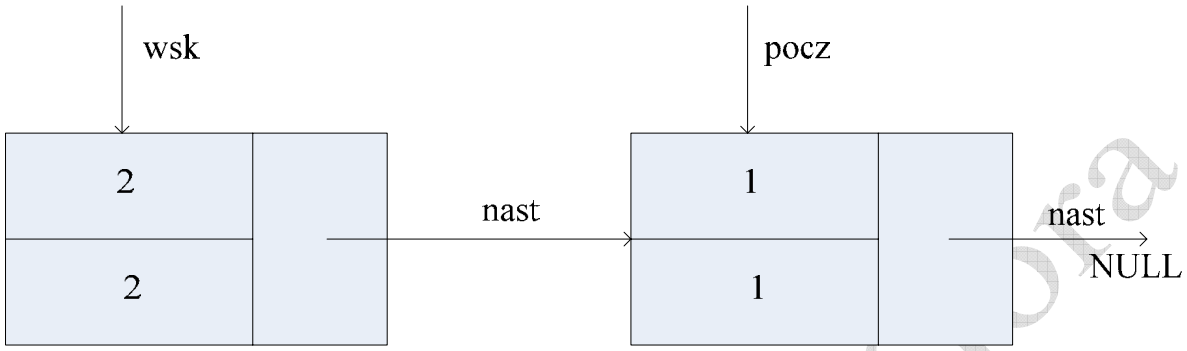
```
wsk->nast=NULL;
wsk->dana_1=2;
wsk->dana_2=2;
```



Rys. 7. Tworzenie listy jednokierunkowej

7) Następuje połączenie elementów listy. Wskaźnik „wsk” pokazywać będzie na element listy który jako pierwszy został zdeklarowany. Wskaźnik „wsk” nie jest przesuwany a jedynie pokazuje na element listy, który jako pierwszy został zdeklarowany na liście.

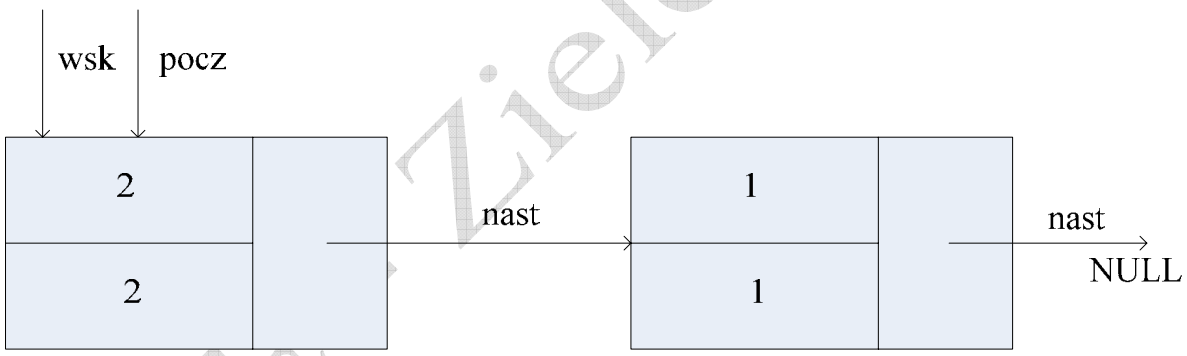
```
wsk->nast=pocz;
```



Rys. 8. Tworzenie listy jednokierunkowej

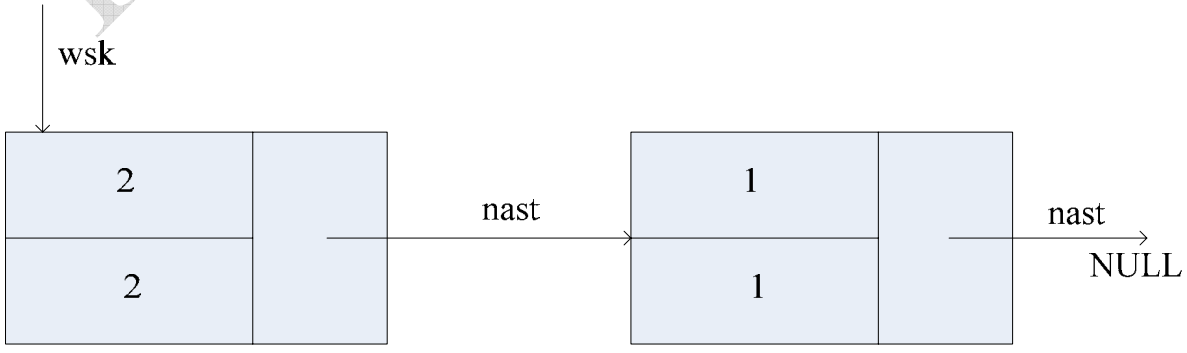
8) Następuje przesunięcie wskaźnika „pocz” tak, że będzie teraz ustawiony i jednocześnie będzie pokazywał na element listy który jako drugi został zadeklarowany.

```
pocz = wsk;
```



Rys. 9. Tworzenie listy jednokierunkowej

9) Proces tworzenia listy jednokierunkowej metodą wstawiania elementów na początku listy został zakończony.



Rys. 10. Zakończenie procesu tworzenia listy jednokierunkowej

Poniżej przedstawiono operację wyświetlenia elementów listy. Po wyświetleniu listy następuje zwolnienie obszaru pamięci zajmowanego przez zmienną „wsk”.

```
while(wsk!=0)
 {
   printf("%d %d\n",wsk->dana_1, wsk->dana_2);
   wsk=wsk->nast;
 }
free(wsk);
```

Uwaga 9!

Przy zapisie „`while(wsk!=0)`” wskaźnik „wsk” przyjmie po zakończeniu działania pętli wartość „0”, a nie ustawi się na ostatnim elemencie listy.

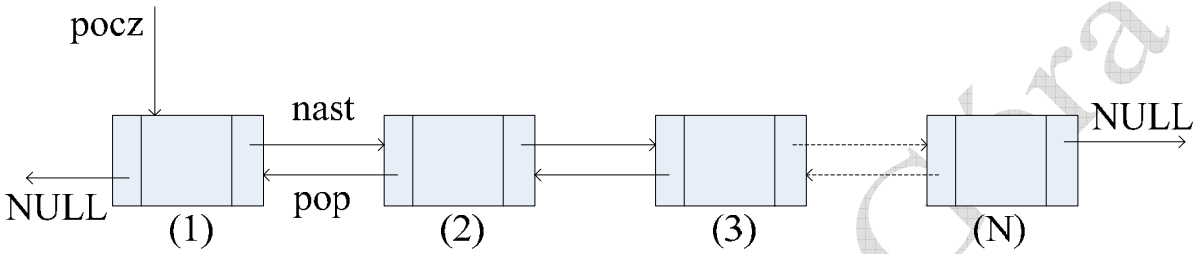
Przykład 88.

W przykładzie następuje wygenerowanie listy jednokierunkowej ze wstawianiem elementów od końca listy.

```
void Push(int liczba){
   struct Element_listy_jednokierunkowej *pocz=NULL,
                                         *wsk=NULL, *pom=NULL;

   pocz=malloc(sizeof(struct Element_listy_jednokierunkowej));
   pocz->dana_1=liczba;
   pocz->dana_2=liczba;
   pocz->nast=NULL;

   wsk=malloc(sizeof(struct Element_listy_jednokierunkowej));
   wsk->dana_1=liczba+1;
   wsk->dana_2=liczba+1;
   wsk->nast=NULL;

   /*wskaznik "pocz" bedzie pokazywal na drugi element listy*/
   pocz->nast=wsk;

   /*wskaznik "pocz" bedzie pokazywal na pierwszy zdeklarowany
                                                 element listy*/
   pom=pocz;

   /*wyswetlenie elementow listy za pomoca petli*/
   while(pom!=0) {
       printf("%d %d\n",pom->dana_1, pom->dana_2);
       pom=pom->nast;
   }
   free(wsk); free(pocz); free(pom);
 }
```

2. Lista dwukierunkowa

Lista dwukierunkowa zawiera pewne dane (elementy listy) oraz *wskaźniki* na następny i poprzedni element listy. Wskaźniki na następny element listy ostatniego elementu listy oraz wskaźnik na poprzedni element listy pierwszego elementu listy mają wartość „NULL". Dostęp do listy umożliwia wskaźnik ustawiony na jej pierwszym elemencie (na *Rys. 1* jest to wskaźnik „*pocz*"). Możliwe jest również zastosowanie wskaźnika ustawionego na ostatnim elemencie listy dwukierunkowej, przeglądanie listy można wówczas wykonywać od przodu lub od tyłu.



Rys. 11. N-elementowa lista dwukierunkowa

Elementy listy dwukierunkowej są *strukturami*. W *języku ANSI C/C++* jeden element listy dwukierunkowej ma następującą postać:

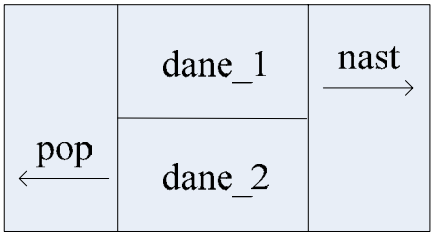
```
struct Lista_dwukierunkowa
{
 typ_danej dana_1;
 ...
 typ_danej dana_n;
 struct Lista_dwukierunkowa *nast, *pop;
};
```

Przykład 89.

W przykładzie następuje wygenerowanie listy dwukierunkowej, ze wstawianiem elementów na początku listy.

1) Następuje zdefiniowanie struktury o dwóch polach „dane_1" i „dane_2" oraz zmiennych wskaźnikowych „nast" i „pop" wskazujących na następny i poprzedni element listy.

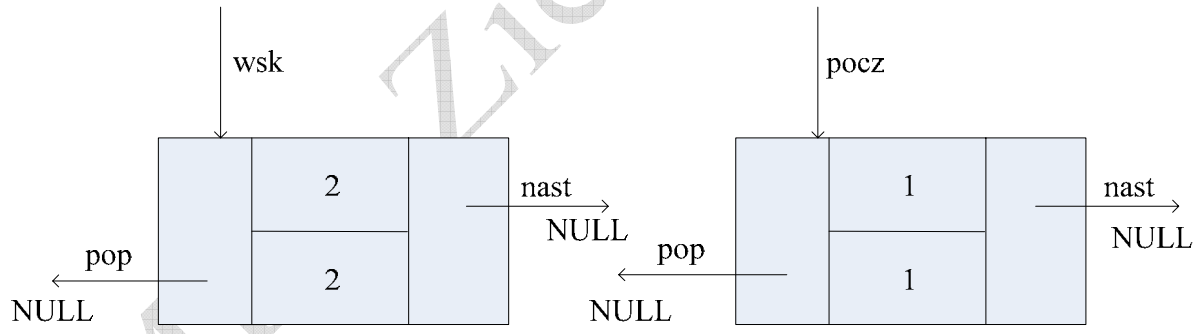
```
struct Element_listy_dwukierunkowej
{
  int dana_1;
  int dana_2;
  struct Element_listy_jednokierunkowej *nast, *pop;
};
```



Rys. 12. Budowa elementu listy dwukierunkowej

2) Następuje deklaracja dwóch wskaźników „pocz” i „wsk” i jednocześnie elementów listy. Elementom listy zostają zadane wszystkie wartości początkowe.

```
struct Element_listy_dwukierunkowej *pocz=NULL, *wsk=NULL;

pocz=malloc(sizeof(struct Element_listy_dwukierunkowej));
pocz->dana_1=1;
pocz->dana_2=1;
pocz->nast=NULL;
pocz->pop=NULL;

wsk=malloc(sizeof(struct Element_listy_dwukierunkowej));
wsk->dana_1=2;
wsk->dana_2=2;
wsk->nast=NULL;
wsk->pop=NULL;
```



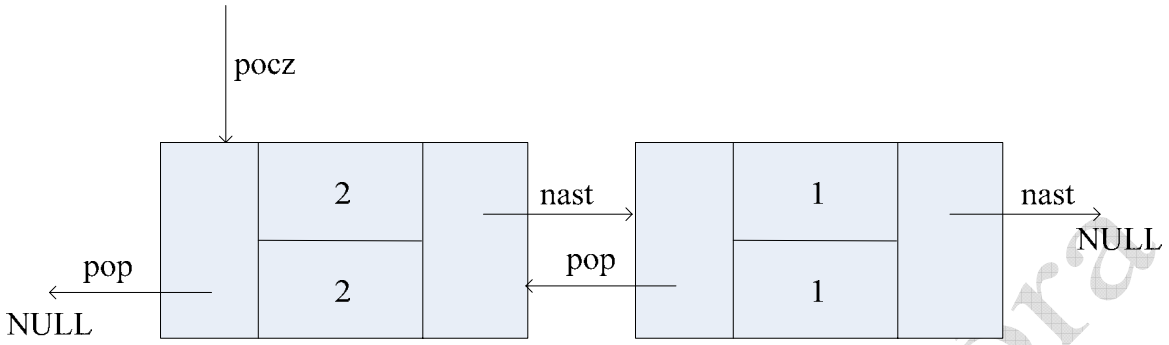
Rys. 13. Budowa listy dwukierunkowej

3) Następuje połączenie elementów listy oraz przesunięcie wskaźnika „pocz” na początek listy. Po wyświetleniu listy wymyty zostaje wskaźnik „wsk”.

```
wsk->nast=pocz;
pocz->pop=wsk;
pocz=wsk;

while(wsk!=0) {
   printf("%d %d\n",wsk->dana_1, wsk->dana_2);
   wsk=wsk->nast;
}
free(wsk);
```

Uwaga 10!

Przy zapisie „`while(wsk!=0)`” wskaźnik „wsk” przyjmuje po zakończeniu działania pętli wartość „0”, a nie ustawi się na ostatnim elemencie listy.



Rys. 14. Budowa listy dwukierunkowej

4) Proces tworzenia listy dwukierunkowej metodą wstawiani elementów na początku listy został zakończony.

Przykład 90.

W przykładzie następuje wygenerowanie listy dwukierunkowej, ze wstawianiem elementów od końca listy.

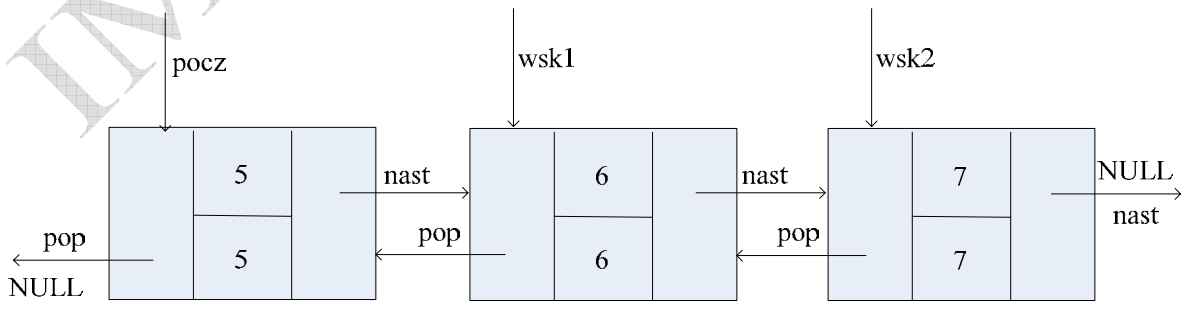
```
void Push(int liczba) {
  struct Element_listy_dwukierunkowej *pocz=NULL, *wsk=NULL;

  pocz=malloc(sizeof(struct Element_listy_dwukierunkowej));
  pocz->dana_1=liczba;   pocz->dana_2=liczba;
  pocz->nast=NULL; pocz->pop=NULL;

  wsk=malloc(sizeof(struct Element_listy_dwukierunkowej));
  wsk->dana_1=liczba+1;   wsk->dana_2=liczba+1;
  wsk->nast=NULL; wsk->pop=NULL;

 /*polaczenie elementow listy (dolaczenie elementu na koncu listy)*/
  pocz->nast=wsk;
  wsk->pop=pocz;

 /*przesuniecie wskanika "wsk" na poczatek listy - konieczne ze
   wzgledu na pozniejsze przegladanie listy*/

  wsk=pocz;

  while(wsk!=0)  {
     printf("%d %d\n",wsk->dana_1, wsk->dana_2);
     wsk=wsk->nast;
  }
  free(wsk);  free(pocz);
}
```

Przykład 91.

W przykładzie przedstawiono sposób usuwania wybranego elementu z 3-elementowej listy dwukierunkowej.

1) Utworzenie 3-elementowej listy dwukierunkowej:

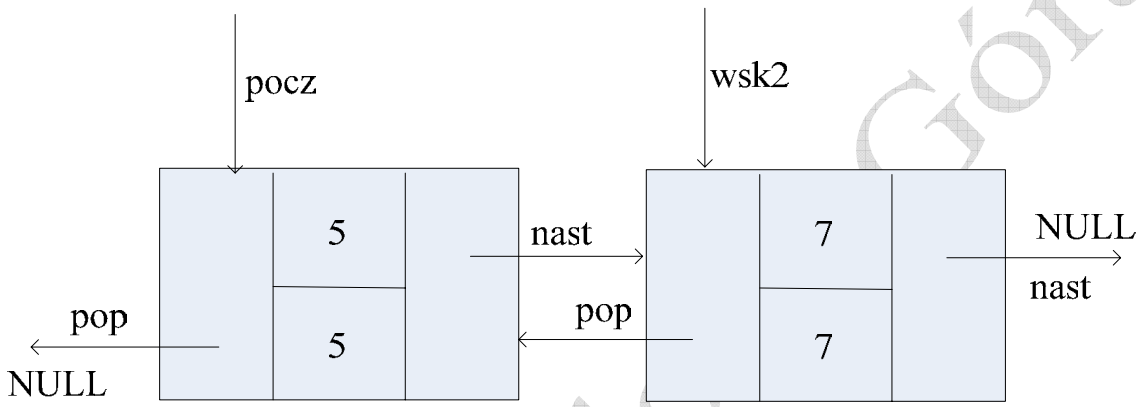
```
/*utworzenie 1-go elementu listy*/
pocz=malloc(sizeof(struct Element_listy_dwukierunkowej));
pocz->dana_1=liczba;
pocz->dana_2=liczba;
pocz->nast=NULL;
pocz->pop=NULL;

/*utworzenie 2-go elementu listy*/
wsk1=malloc(sizeof(struct Element_listy_dwukierunkowej));
wsk1->dana_1=liczba+1;
wsk1->dana_2=liczba+1;
wsk1->nast=NULL;
wsk1->pop=NULL;

/*utworzenie 3-go elementu listy*/
wsk2=malloc(sizeof(struct Element_listy_dwukierunkowej));
wsk2->dana_1=liczba+2;
wsk2->dana_2=liczba+2;
wsk2->nast=NULL;
wsk2->pop=NULL;

/*polaczenie elementow listy (dolaczenie elementów na koncu
  listy)*/
pocz->nast=wsk1;
wsk1->pop=pocz;

wsk1->nast=wsk2;
wsk2->pop=wsk1;
```



Rys. 15. Budowa 3-elementowej listy dwukierunkowej

2) Usunięcie 2-go elementu listy, połączenie 1-go i trzeciego elementu listy:

```
wsk1->nast=NULL;
wsk1->pop=NULL;
pocz->nast=NULL;
wsk2->pop=NULL;

free(wsk1);

pocz->nast=wsk2;
wsk2->pop=pocz;
```

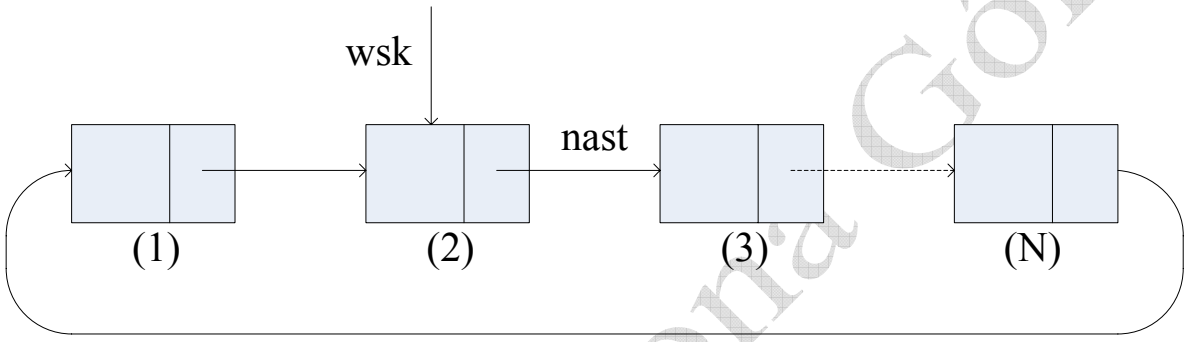


Rys. 16. Usuniecie 2-go elementu listy dwukierunkowej

3. Lista cykliczna

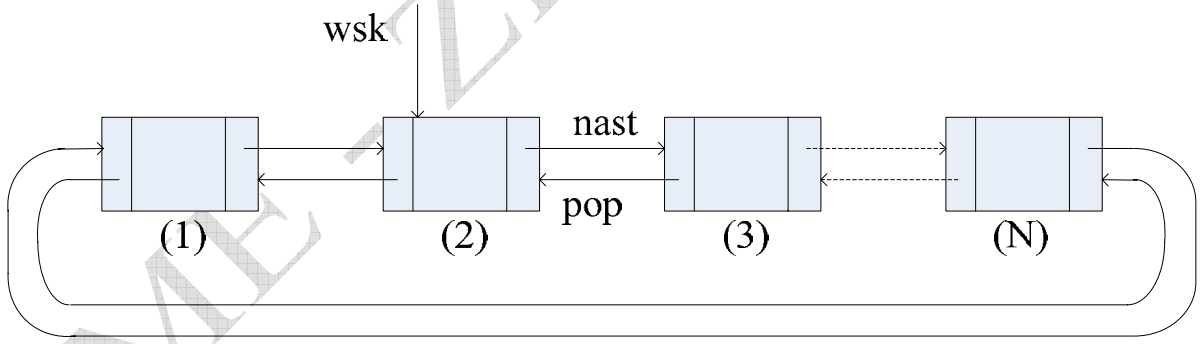
Wyróżniamy *listy cykliczne* jednokierunkowe i dwukierunkowe. W listach cyklicznych wskaźnik na następny element w ostatnim elemencie nie ma wartości NULL, ale wskazuje na początek listy. W przypadku list cyklicznych dwukierunkowych, wskaźnik na element poprzedni w pierwszym elemencie listy wskazuje na element ostatni, zaś wskaźnik elementu ostatniego listy pokazuje na pierwszy element tej listy (Rys.18). Listy cykliczne nie posiadają początku i końca. Pozycję na liście określa zewnętrzny wskaźnik, którego położenie może się dowolnie zmieniać. Struktury w *języku ANSI C/C++* odpowiadające elementom list cyklicznych są takie same jak odpowiadających im list zwykłych.

Struktura logiczna listy cyklicznej jednokierunkowej:



Rys. 17. Struktura listy cyklicznej jednokierunkowej

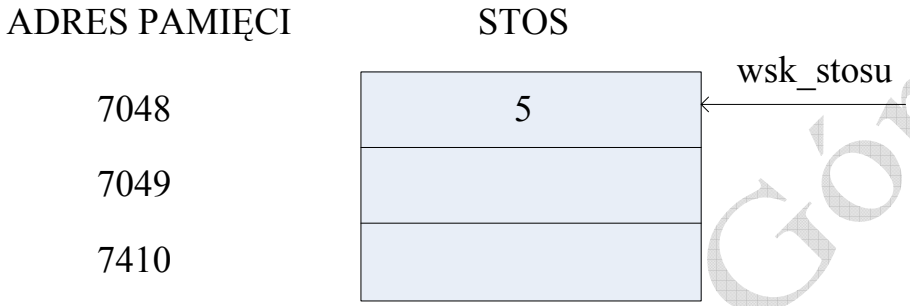
Struktura logiczna listy cyklicznej dwukierunkowej:



Rys. 18. Struktura listy cyklicznej dwukierunkowej

4. Stos

Stos jest strukturą często używaną w programowaniu. Stos bywa również nazywany kolejką LIFO (Last In First Out). Elementy ostatnio położone na stosie są z niego zdejmowane przed elementami położonymi wcześniej. Do obsługi stosu służy zmienna nazywana *wskaźnikiem stosu* (na *Rys. 19* jest to „wsk_stosu"). Wskaźnik stosu może pokazywać ostatnio położony element lub pierwsze wolne miejsce na stosie. Ze stosu zdejmowany jest element, na który pokazuje wskaźnik stosu.



Rys. 19. Struktura stosu

Stos może być implementowany za pomocą tablicy jednowymiarowej. Wskaźnikiem stosu jest wtedy zmienna będąca indeksem w tablicy wskazująca ostatnio położony na stosie element. Stos musi zapewniać poprawne wykonanie przynajmniej dwóch funkcji: *Push (element)* - położenie elementu na stosie, *Pop()* - zdjęcie elementu z wierzchołka stosu.

Stos może być również implementowany przy użyciu listy jednokierunkowej. Szczyt stosu znajduje się wtedy na początku listy. Nowe elementy kładzione na stos są zawsze dołączane na początek listy. Elementy zdejmowane są zawsze z początku (wierzchołka) stosu.

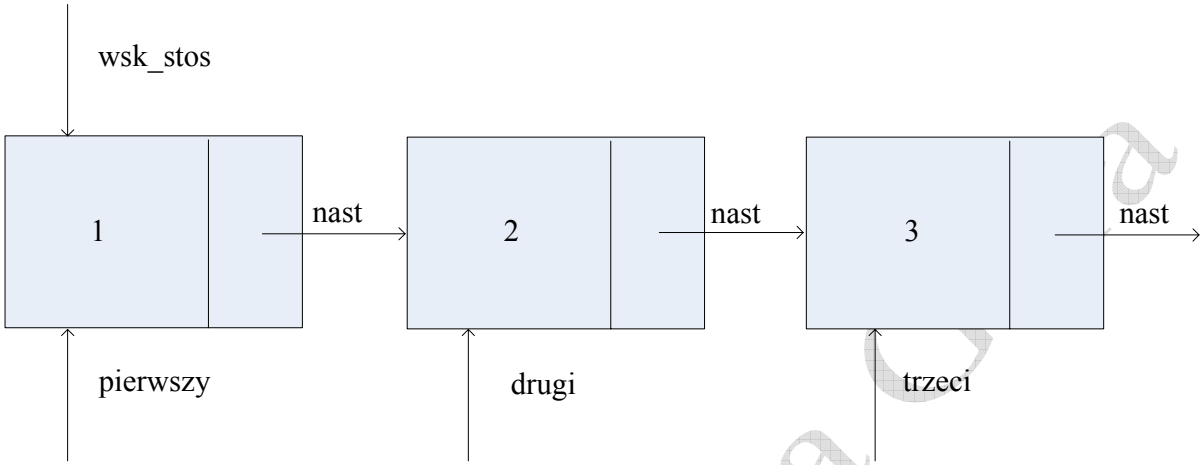
Przykład 92.

W przykładzie pokazano zastosowanie listy jednokierunkowej do zaimplementowania stosu.

1) Następuje wygenerowanie stosu oraz ustawienie wskaźnika „wsk_stos" na wierzchołek stosu.

```
pierwszy=malloc(sizeof(struct Element_listy_jednokierunkowej));
pierwszy->dana=1;
pierwszy->nast=NULL;

drugi=malloc(sizeof(struct Element_listy_jednokierunkowej));
drugi->dana=2;
drugi->nast=NULL;

trzeci=malloc(sizeof(struct Element_listy_jednokierunkowej));
trzeci->dana=3;
trzeci->nast=NULL;
```

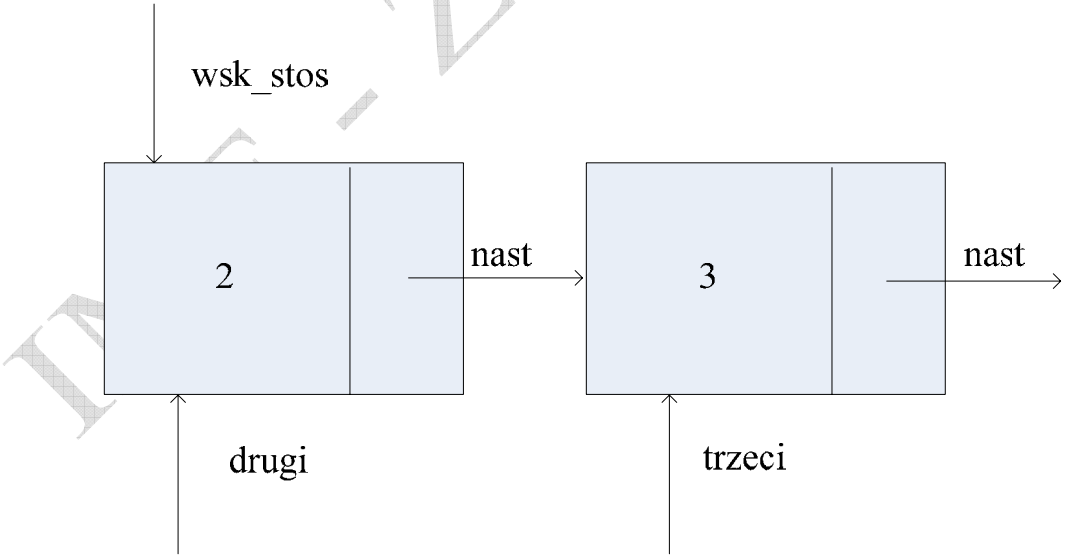
```
/*polaczenie elementow stosu*/
 pierwszy->nast=drugi;
 drugi->nast=trzeci;

/*ustawienie wskaznika "wsk_stos" na wierzcholek stosu*/
 wsk_stos=pierwszy;
```



Rys. 20. Organizacja 3-elementowego stosu

2) Zdjęcie elementu ze stosu (*pop()*).

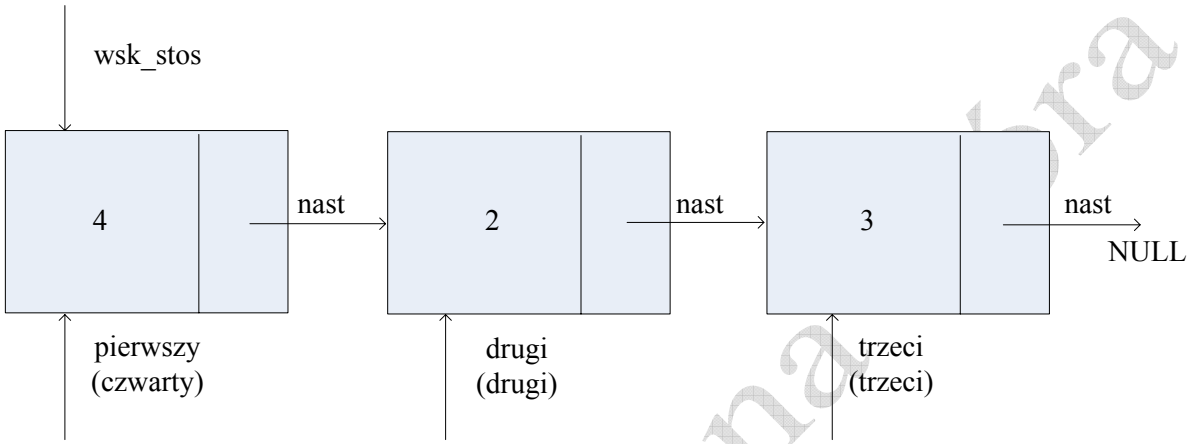
```
wsk_stos=drugi; //drugi z elementow jest teraz na wierzcholku stosu
pierwszy->nast=NULL;
free(pierwszy);
```



Rys. 21. Zdjęcie elementu ze stosu

3) Położenie elementu na stosie(*push(element)).*

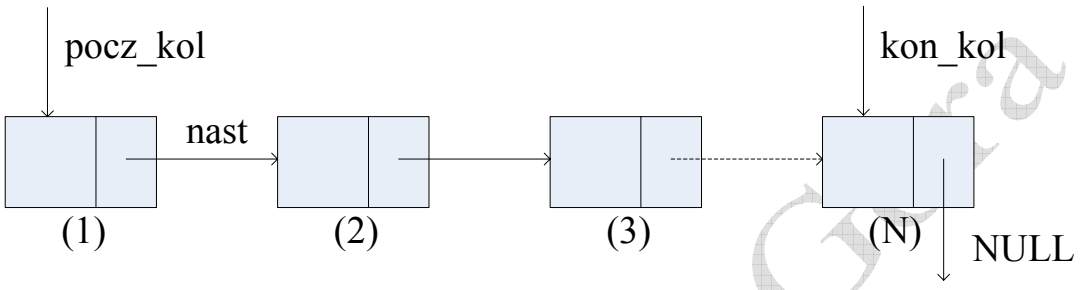
```
czwarty=malloc(sizeof(struct Element_listy_jednokierunkowej));
czwarty->dana=4;
czwarty->nast=NULL;

czwarty->nast=drugi;
wsk_stos=czwarty; //przesuniecie wskaznika "wsk_stos" na
                  //wierzcholek stosu
```



Rys. 22. Wstawienie elementu na stos

5. Kolejka

Kolejka jest dynamiczną strukturą danych typu FIFO (First In, First Out). Kolejka może być implementowana na liście jednokierunkowej przy pomocy dwóch wskaźników. Na *Rys. 23* wskaźnik „pocz_kol" pokazuje na początek listy (jest to równocześnie wskaźnik na pierwszy element, który zostanie z kolejki pobrany) natomiast wskaźnik „kon_kol" pokazuje na koniec listy (jest to miejsce, gdzie będą wstawiane elementy dopisywane do kolejki).



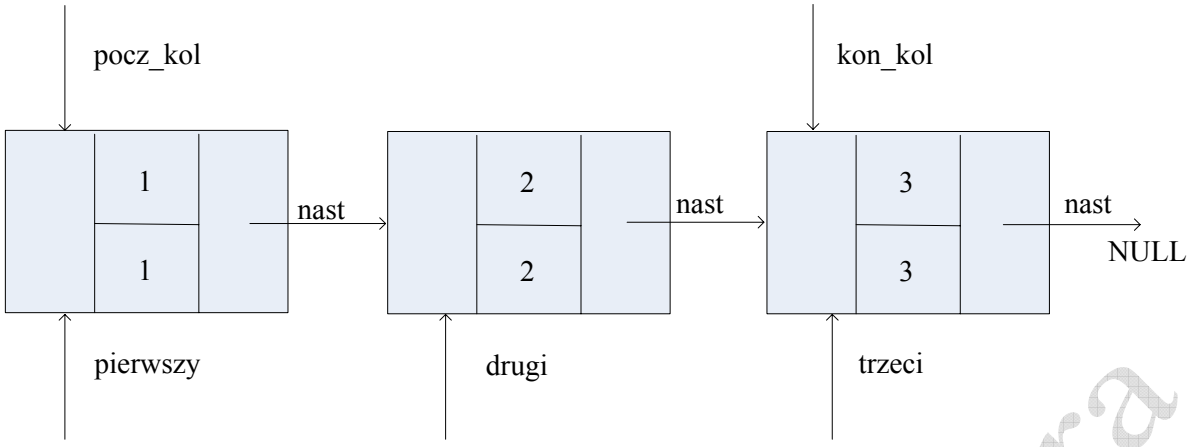
Rys. 23. Organizacja kolejki

Przykład 93.

W przykładzie pokazano zastosowanie listy jednokierunkowej do zaimplementowania kolejki.

1) Następuje wygenerowanie i połączenie elementów kolejki oraz ustawienie wskaźników „pocz_kol" i „kon_kol" na początek i koniec kolejki.

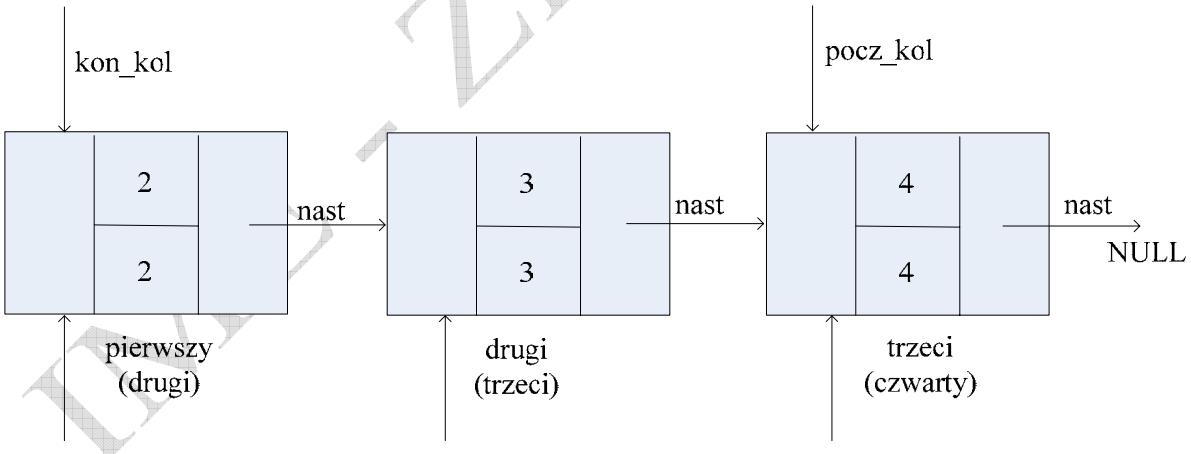
```
 pierwszy=malloc(sizeof(struct Element_listy_jednokierunkowej));
 pierwszy->dana_1=1;
 pierwszy->dana_2=1;
 pierwszy->nast=NULL;

 drugi=malloc(sizeof(struct Element_listy_jednokierunkowej));
 drugi->dana_1=2;
 drugi->dana_2=2;
 drugi->nast=NULL;

 trzeci=malloc(sizeof(struct Element_listy_jednokierunkowej));
 trzeci->dana_1=3;
 trzeci->dana_2=3;
 trzeci->nast=NULL;

/*polaczenie elementow kolejki*/
 pierwszy->nast=drugi;
 drugi->nast=trzeci;

/*ustawienie wskaznikow "pocz_kol" i "kon_kol" na poczatek i koniec
                                                          kolejki*/
 pocz_kol=pierwszy;
 kon_kol=trzeci;
```



Rys. 24. Organizacja 3-elementowej kolejki

2) Następuje zdjęcie pierwszego elementu z kolejki (zwolnienie wskaźników, przesunięcie „pocz_kol" na drugi element kolejki oraz wymycie pierwszego elementu kolejki) i dostawienie nowego elementu na koniec kolejki. Drugi element starej kolejki staje się pierwszym elementem nowej kolejki.

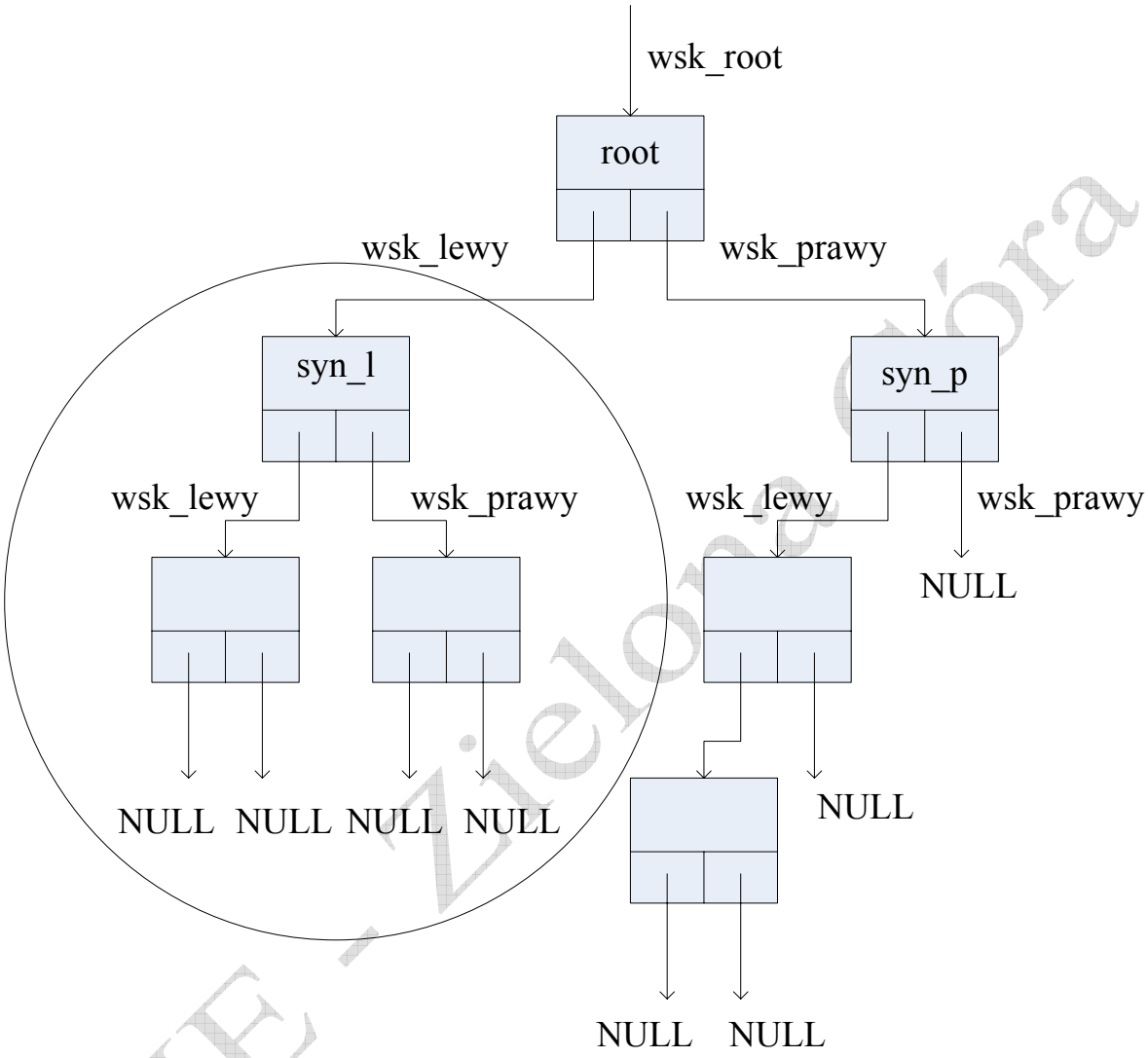
```
czwarty=malloc(sizeof(struct Element_listy_jednokierunkowej));
czwarty->dana_1=4; czwarty->dana_2=4;
czwarty->nast=NULL;

pocz_kol=drugi;   //drugi z elementow jest teraz pierwszy w kolejce
pierwszy->nast=NULL;
free(pierwszy);
trzeci->nast=czwarty; //na koncu kolejki dochodzi nowy element
```



Rys. 25. Organizacja 3-elementowej kolejki, dojście nowego elementu

6. Drzewa

Drzewa są strukturami hierarchicznymi i rekurencyjnymi. Można je podzielić na drzewa mnogościowe (każdy węzeł może mieć dowolną ilość potomków) lub drzewa binarne, gdzie każdy węzeł ma nie więcej niż dwóch potomków.



Rys. 26. Struktura typowego drzewa binarnego

Wskaźnik (wsk_root) umożliwiający dostęp do całego drzewa wskazuje na węzeł, który nie jest niczyim potomkiem. Węzeł ten nazywa się korzeniem drzewa (root). Każdy węzeł drzewa binarnego ma co najwyżej dwóch potomków - synów: lewego (syn_l) i prawego (syn_p). Jeśli pewien węzeł nie ma jednego z potomków to odpowiedni wskaźnik jest ustawiony na NULL.

Rekurencyjność struktury drzewiastej polega na tym, że każdy węzeł może być rozpatrywany jako drzewo (część zaznaczona w kółku może być potomkiem korzenia), ale również może być traktowany tak jak osobne drzewo binarne. Stąd algorytmy operujące na strukturach drzewiastych są najczęściej funkcjami rekurencyjnymi.

Elementy drzewa są *strukturami*. W *języku ANSI C/C++* jeden element drzewa ma następującą postać:

```
struct Elementy_drzewa {
  typ_danej dana_1;
  ...
  typ_danej dana_n;
  struct Elementy_drzewa *wsk_lewy, *wsk_prawy;
};
```

Drzewa binarne stosowane są do przechowywania danych. Dana przechowywane są w węzłach drzewa. Pierwsza informacja jest wstawiana jako korzeń drzewa. Kolejne dane są wstawiane tak, że w każdym węźle dane mniejsze lub równe od danej w tym węźle znajdują się na lewo, natomiast większe - na prawo. Taka organizacja przyspiesza znacznie wyszukiwanie elementów zapisanych w drzewie, ponieważ chcąc sprawdzić czy dana informacja istnieje, nie sprawdza się wszystkich elementów, ale tylko jedną ścieżkę w drzewie.

Przykład 94.

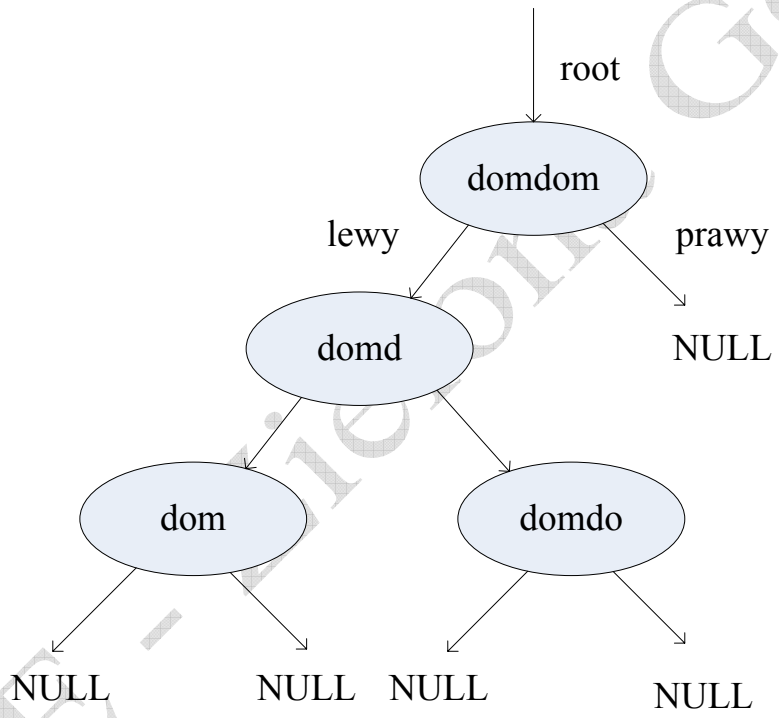
W przykładzie przedstawiono funkcję tworzącą drzewo binarne oraz funkcję kasującą drzewo binarne.

```
struct drzewo {
 char dana[100];
 struct drzewo *lewy, *prawy;
};
```

```
/*wstawianie elementow do drzewa, w celu przechowania historii
dodawanych elementow stosuje się wskaznik na wskaznik*/
void f_puts_string(struct drzewo **root, char * name)
{
 if (*root == NULL) {
   *root = malloc(sizeof(struct dzewo));

    if (* root == NULL)   {
     printf("Blad przydzialu pamieci!\n");
     exit(1);
    }
    /*wpisanie danych do kolejnego korzenia*/
    strcpy((* root) -> dana, name);
    (* root) -> lewy = (* root) -> prawy = NULL;
 }
 /*porownanie wprowadzonych napisow, dluzszy od poprzedniego
   dorzucony bedzie na prawo od swojego korzenia, krotszy na lewo*/
 if (strcmp((*root) -> dana, name) > 0)
    f_puts_string(& (*root) -> prawy, name); //rekurencja dla
                                             //kazdego liscia ktory
                                             //jest jednoczesnie
                                             //korzeniem
 else f_puts_string(& (*root) -> left, name);
}
```

```
/*kasowanie drzewa*/

void Usun_drzewo(struct drzewo *root)
{
 if (root == NULL) return;
 Usun_drzewo (root -> lewy); //rekurencja w celu ustalenia
                             //polozenia elementu w drzewie
 Usun_drzewo (root -> prawy;
 free(root);
}
```

Np. Wprowadzamy kolejno słowa {„domdom”, „domd”, „dom”, „domdo” } wówczas drzewo binarne wprowadzonych słów będzie miało następującą postać:



Rys. 27. Przykład drzewa binarnego słów