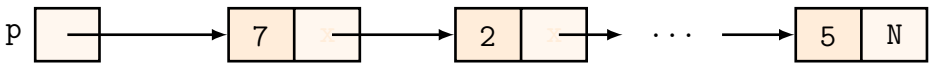
Lista zadań nr 4

Zbiory dynamiczne, unie, wskaźniki do funkcji.

Zadania podstawowe:

Zadanie 1 *Stos* to liniowa struktura danych, w której dane dokładane są na wierzch stosu i z wierzchołka stosu są pobierane. Stos może zostać zaimplementowany jako ograniczona wersja listy jednokierunkowej. Nowe węzły mogą być dodawane do stosu i usuwane jedynie z wierzchu stosu. Dlatego stos jest określany mianem struktury typu LIFO (ang. *last in, first out*, czyli ostatni na wejściu, pierwszy na wyjściu). Dostęp do stosu odbywa się za pomocą wskaźnika prowadzącego do elementu znajdującego się na wierzchu stosu. Element składowy łącza w ostatnim węźle stosu ma przypisaną wartość `NULL` wskazującą, że dany element znajduje się n samym dole stosu. Rysunek 2 przedstawia przykład stosu z kilkoma węzłami - `p` prowadzi do elementu znajdującego się na wierzchu stosu. Przedstawiony tutaj stos i lista przedstawiona na wykładzie wyglądają identycznie. Rożnica między tymi strukturami polega na tym, że operacja wstawiania i usuwania węzłów można przeprowadzać gdziekolwiek na liście jednokierunkowej, natomiast w przypadku stosu tylko na jego wierzchu.



Rys. 1: Graficzne przedstawienie struktury stosu.

Podstawowymi funkcjami używanymi do przeprowadzania operacji na stosie są `push()` i `pop()`. Funkcja `push()` powoduje utworzeni nowego węzła i jego umieszczenie na wierzchu stosu. Funkcja `pop()` natomiast usuwa węzeł z wierzch stosu, zwalani pamięć zaalokowaną dla usuniętego węzła i zwraca usuniętą wartość.

Napisz program, który będzie zawierał prostą implementację stosu. Program powinie oferować trzy opcje: umieszczanie wartości na stosie - opcja 1, która wywołuje funkcję `push()`; usunięcie wartości ze stosu - opcja 2, która wywołuje funkcję `pop()`; zakończenie działania programu - opcja 3. Po każdorazowym wybraniu opcji 1 lub 2 program powinien wyświetlać aktualny stan stosu - zdefiniuj funkcję `printStack()`.

Zadanie 2 Stos możemy również zaimplementować za pomocą tablicy dynamicznej. Napisać funkcje pozwalające reprezentować stos za pomocą tablicy. Pamięć dla tablicy ma być przydzielana dynamicznie. W przypadku, gdy tablica jest pełna, funkcja wstawiająca wartość na stos powinna zwiększyć dwukrotnie jej rozmiar (wykorzystaj funkcję `realloc()`). Jeśli po usunięciu wartości ze stosu, stos zajmuje nie więcej niż jedną czwartą wszystkich elementów tablicy, to rozmiar tablicy należy zmniejszyć dwukrotnie.

Zadanie 3 W pliku *labirynt.txt* zapisano mapę reprezentującą labirynt. Mapa ma kształt prostokąta o wymiarach `10`×`18` i jest podzielonego na jednostkowe kwadratowe pola. Mapa zawiera `10` wierszy, z których każdy ma po `18` znaków opisujących pola. Znak `@` oznacza, że odpowiednie pole jest puste, a znak `X` – przeszkodę terenową (np. skała). Przykładowy opis mapy labiryntu może wyglądać następująco:

```
XXXXXXXXXXXXXXXXXX
@@X@@@@@@@@@@@@@XX
X@X@X@XXX@XXX@XXXX
X@X@X@XXX@X@@@@@@X
X@X@X@XXX@@XX@XXXX
X@X@XXXXXX@XXXXXXX
X@@@@@@@XX@@@@@@@X
X@XXXXXXXX@XX@X@XX
X@@@@X@@@@@XX@X@@@
XXXXXXXXXXXXXXXXXX
```

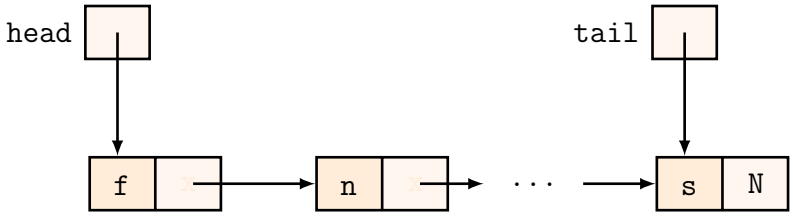
Wejście do labiryntu znajduje się w jednym polu oznaczonym znakiem `@` znajdującym się pierwszej kolumnie, a wyjście w ostatniej kolumnie w jednym polu ze znakiem `@`.

Zadanie polega na napisaniu programu, który wyznacza ścieżkę przejścia przez labirynt z wykorzystaniem strategii przeszukiwania w głąb (listę kolejnych pól, które należy odwiedzić w czasie przechodzenia po labiryncie). Przechodząc przez labirynt wybieramy kierunki przemieszczania się według zasady, że gdy mamy możliwość wyboru kierunku najpierw wybieramy wschód, później południe, zachód, a na koniec sprawdzamy ścieżkę w kierunku północnym. Jeżeli dojdziemy do "ślepej uliczki" cofamy się do miejsca gdzie jest jeszcze możliwość wyboru i badamy kolejny kierunek zgodnie z opisaną regułą.

Podczas pisania programu pomocne może być wykorzystanie stosu (wykorzystaj implementację z zadania 1 lub 2). Sam labirynt należy reprezentować przez dwuwymiarową tablicę znaków. Stosu możemy używać, aby śledzić pola wzdłuż swojej bieżącej ścieżki. Za każdym razem, gdy przechodzimy do nowego pola, wpisujemy je na stos reprezentujący bieżącą ścieżką. Gdy się cofamy, zdejmujemy pole ze stosu i wracamy do tego pola, które było przed nim. W ten sposób, zawsze wiemy, jak się wycofywać. Dane na stosie (współrzędne pola) mogą być reprezentowane przez odpowiednia strukturę.

Zadanie 4 Inną, dość prostą i często stosowaną strukturą danych jest *kolejka*. Ta struktura jest podobna do kolejki, którą można zobaczyć na przykład w sklepie: pierwsza osoba w kolejce zostanie obsłużona na początku, pozostali mogą stawać na końcu kolejki i muszą czekać na obsłużenie. Węzeł kolejki jest usuwany tylko na początku kolejki, natomiast wstawiany tylko na końcu kolejki. Dantego kolejka jest określana mianem struktury danych typu FIFO (ang. *first in, first out*, czyli pierwszy na wejściu, pierwszy na wyjściu). Operacje wstawiania i usu-

wania są nazywane odpowiednio enqueue (z ang. kolejkuj) i dequeue (z ang. odkolejkuj). Rysunek



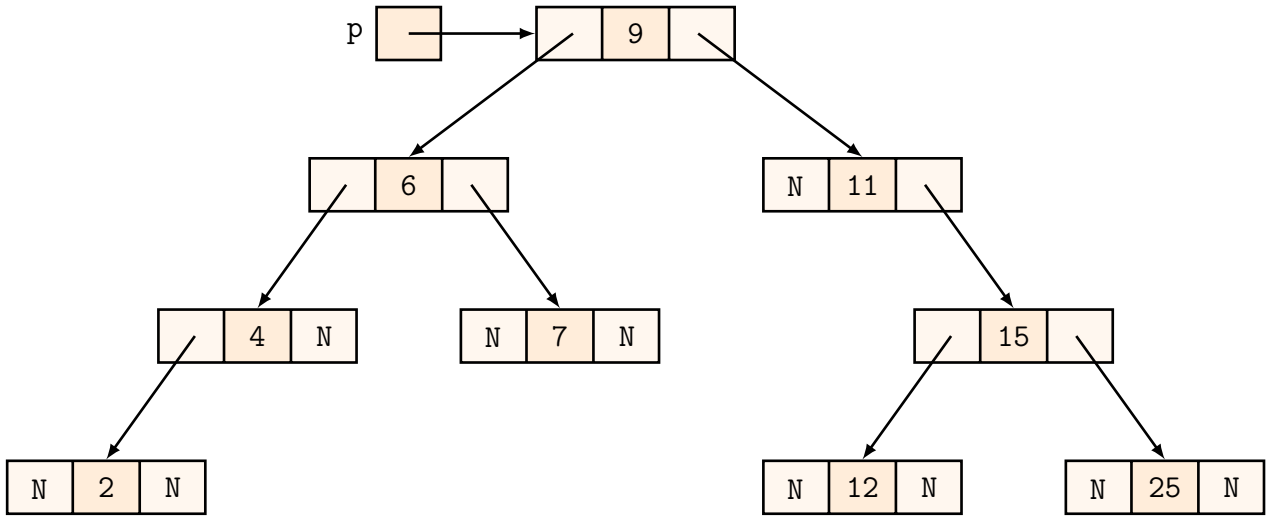
Rys. 2: Graficzne przedstawienie struktury kolejki.

Napisz program, który będzie zawierał implementację prostej kolejki. Program powinien oferować kilka opcji: wstawianie węzła do kolejki (wywołanie funkcji `enqueue()`), usunięcie węzła z kolejki (wywołanie funkcji `dequeue()`) i zakończenie działanie programu. Po każdorazowym wybraniu opcji usuwania lub wstawiania węzła do kolejki program powonień wyświetlać aktualny stan kolejki - zdefiniuj funkcję `printQueue()`.

Zadanie 5 Każda ze struktur typu

```
struct node{
    int data;
    stuct node *left, *right;
};
```

zawiera dwa wskaźniki, które mogą wskazywać dwie inne takie struktury. Można z nich utworzyć coś takiego (`N` oznacza wskaźnik pusty `NULL`):



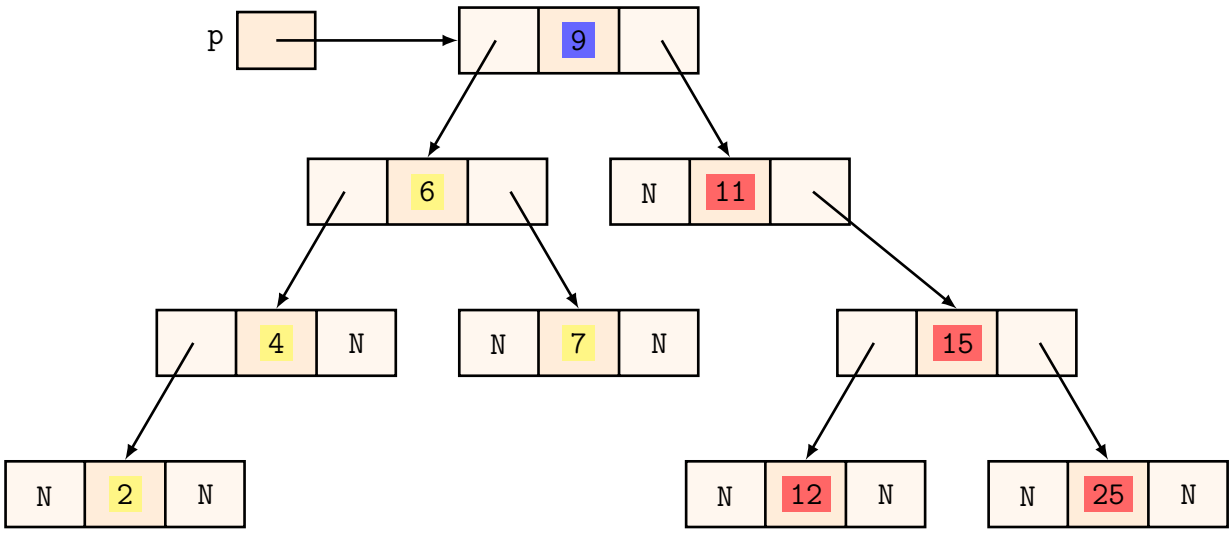
Jedna ze struktur jest wskazywana przez wskaźnik `p`. Każda z pozostałych struktur jest wskazywana przez inną strukturę (jedną). Każdy ze wskaźników będących składową struk-

tury albo wskazuje którąś ze struktur albo jest wskaźnikiem pustym. Na potrzeby tego zadania[1] wprowadzimy następującą terminologię:

(i) Strukturami znajdującymi się po lewej stronie struktury będziemy nazywali strukturę wskazywaną przez wskaźnik lewy tej struktury i wszystkie struktury, do których można "dojść" przechodząc dalej "po strzałkach w dół".

(ii) Strukturami znajdującymi się po lewej stronie struktury będziemy nazywali strukturę wskazywaną przez wskaźnik prawy tej struktury i wszystkie struktury, do których można "dojść" przechodząc dalej "po strzałkach w dół".

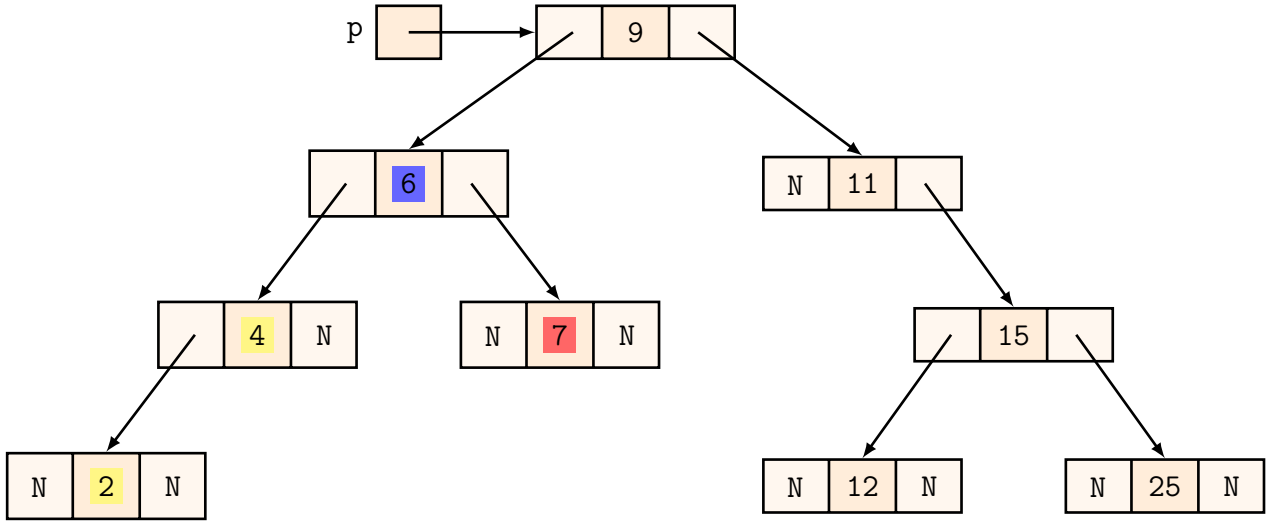
Na poniższych rysunkach struktury z wartościami zaznaczonymi kolorem żółtym znajdują się po lewej stronie struktury, której wartość została zaznaczona kolorem niebieskim, a struktury z wartościami zaznaczonymi kolorem żółtym znajdują się po prawej stronie tej struktury.



Przyjmujemy, że wartości składowej `data` zostały dobrane w taki sposób, by wartości znajdującej się w każdej ze struktur były większe od wartości umieszczonych "po lewej stronie" i mniejsze od wartości umieszczonych "po prawej stronie" (jak na rysunku).

Napisać funkcję, która rekurencyjnie wyświetla wartości składowej dane w kolejności od najmniejszej do największej. Parametrem funkcji ma być wskaźnik wskazujący strukturę. Funkcja wyświetla w kolejności od najmniejszej do największej wartości składowej dane dla struktury wskazywanej przez jej argument oraz struktur znajdujących się po jej "lewej i prawej stronie". Jeśli argument funkcji jest wskaźnikiem pustym, to funkcja nie powinna niczego robić. Jeśli argument funkcji jest niepusty, to powinien wskazywać jedną ze struktur. Należy najpierw wyświetlić wartości mniejsze od wartości składowej dane struktury wskazywanej

[1] Struktury "połączone" wskaźnikami w opisany wcześniej sposób reprezentują tzw. drzewo binarne. Można mówić o lewym i prawym poddrzewie węzła.



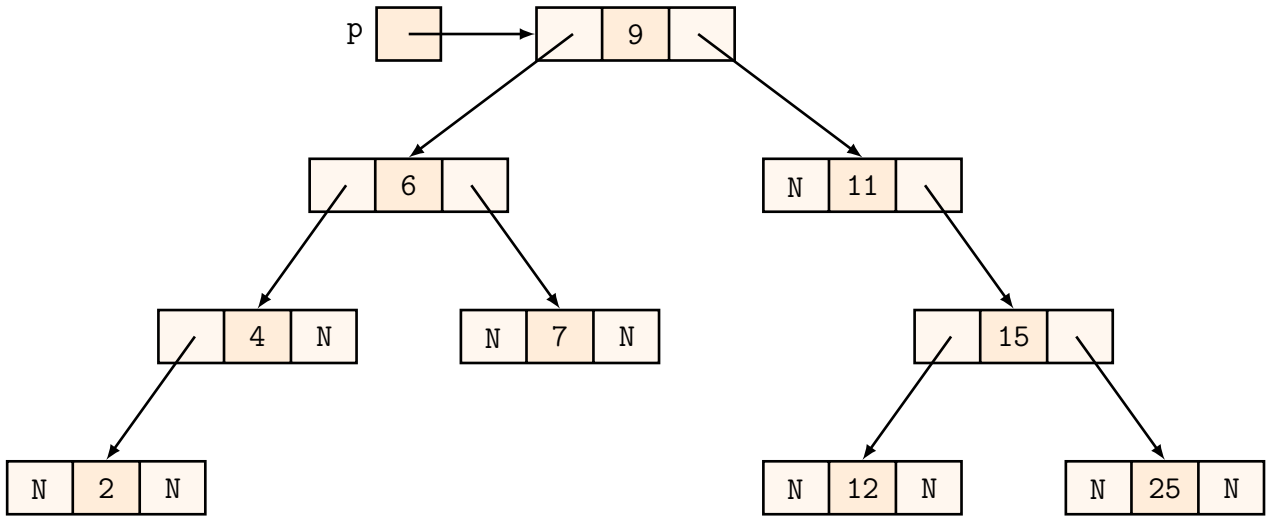
przez argument funkcji (znajdujące się "po lewej stronie" tej struktury). Można to zrobić wywołując funkcję rekurencyjnie i podając jako jej argument wskaźnik lewy. Następnie funkcja powinna wyświetlić wartość składowej dane struktury wskazywanej przez argument funkcji, a potem wartości od niej większe wywołując rekurencyjnie funkcję dla wskaźnika prawy.

Napisać program wyświetlający wartości składowej dane w kolejności od najmniejszej do największej.

Zadanie 6 W rozwiązaniu tego zadania może pomóc wcześniejsze zaznajomienie się z drzewami BST. Rozważamy drzewa utworzone ze struktur

```
struct node{
    int data;
    stuct node *left, *right;
};
```

podobne do drzewa z zadania 5:

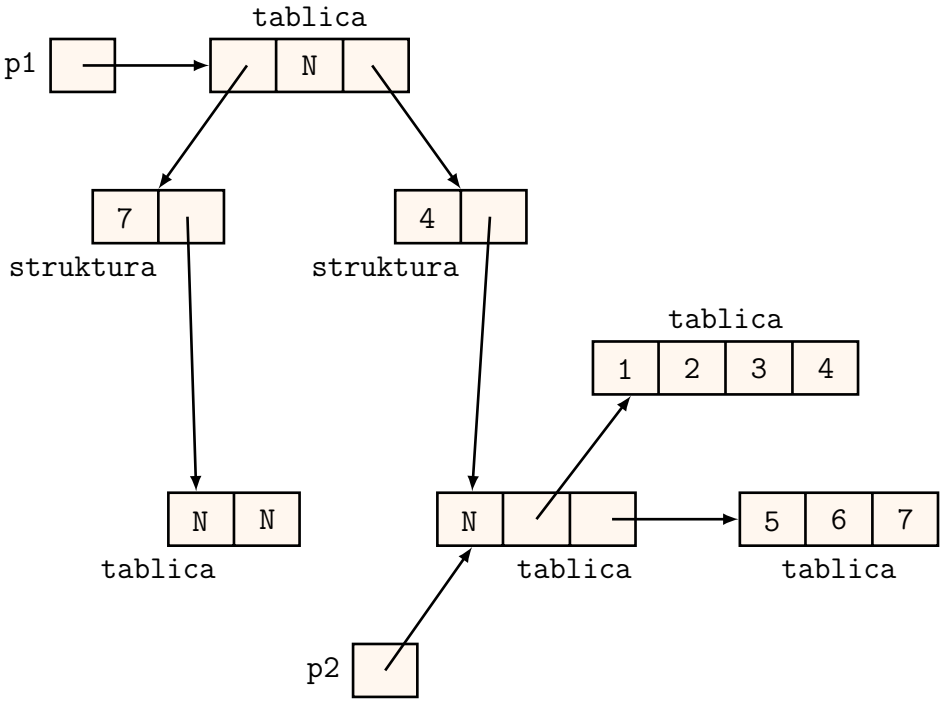


(i) Napisać funkcję wyszukującą strukturę zawierającą zadaną wartość. Przyjąć, że wskaźnik `p` (wskazujący strukturę reprezentującą "korzeń" drzewa) jest zmienną zewnętrzną (globalną). Funkcja ma posiadać jeden parametr służący do podania szukanej wartości. Funkcja powinna zwracać wskaźnik wskazujący strukturę zawierającą poszukiwaną wartość lub wskaźnik pusty w przypadku, gdy wartość nie została odnaleziona. Nie chodzi o to by funkcja przeszukała wszystkie struktury. Należy wykorzystać sposób, w jaki wartości są rozmieszczone ("po lewej stronie" wartości mniejsze, "po prawej stronie" wartości większe). Rozwiązując to zadanie nie należy też korzystać z rekurencji. Wskazówka: Do poszukiwania struktury zawierającej podaną wartość użyć wskaźnik (przyjmiemy, że ten wskaźnik nazywa się `q`). Wskaźnik `q` powinien początkowo wskazywać strukturę wskazywaną przez wskaźnik `p`. Następnie jeśli szukana wartość jest mniejsza od wartości znajdującej się w strukturze wskazywanej przez wskaźnik `q`, to wskaźnik `q` należy "przesunąć w lewo", a jeśli jest większa to "w prawo". "Przesuwanie" wskaźnika `q` ma być kontynuowane do momentu gdy zostanie odnaleziona poszukiwana wartość, lub gdy będzie wiadomo, że żadna ze struktur nie zawiera tej wartości (wskaźnik przyjmie wartość NULL).

(ii) Napisać funkcję dodającą do drzewa jedną wartość. Dodawana wartość ma być podawana jako argument funkcji. Jeśli drzewo nie zawiera żadnej wartości (`p` jest wskaźnikiem pustym), to funkcja powinna przydzielić pamięć dla struktury za pomocą funkcji `malloc()`, następnie umieścić w tej strukturze dodawaną wartość i przypisać adres struktury wskaźnikowi `p`. Jeśli drzewo zawiera już jakieś wartości, to postępowanie powinno być podobne do opisanego w (i). Należy użyć dwa wskaźniki. Przed "przesunięciem" wskaźnika `q` jego wartość powinna zostać przypisana drugiemu wskaźnikowi. Jeśli dodawana wartość już znajduje się w drzewie, to po jej odnalezieniu funkcja przerywa działanie. Jeśli "przesuwany" wskaźnik `q` przyjmie wartość NULL, to jego poprzednia wartość (znajdująca się w drugim wskaźniku) wskaże strukturę, do której należy "doczepić" strukturę z dodawaną wartością (jeśli do pokazanego na rysunku drzewa zostanie dodana struktura zawierająca wartość `3`, to będzie ona wskazywana przez prawy wskaźnik struktury zawierającej wartość `2`).

Zadanie 7 Napisać program, w którym pamięć dla przedstawionych na rysunku 3 tablic i struktur jest przydzielana dynamicznie (funkcją `malloc()`). Program powinien przypisać pokazane na rysunku wartości wskaźnikom `p1` i `p2` oraz tablicom i strukturom (`N` oznacza wskaźnik pusty `NULL`). Dobrać odpowiednio typy, tak by podczas kompilacji programu nie było komunikatów o błędach i program działał poprawnie. Odwołując się do wartości umieszczonych w tablicach i strukturach za pomocą wskaźników `p1` oraz `p2` sprawdzić, czy są one takie same jak na rysunku. W programie można użyć dodatkowe wskaźniki (poza przedsta-

wionymi na rysunku).



Rys. 3: Przykładowa struktura danych.

Zadanie 8 Napisać program, obliczający sumę dwu liczb podanych jako argumenty jego wywołania. Jeśli plik wykonywalny został nazwany `suma` (`suma.exe` w systemie Windows), to po wydaniu polecenia:

```
suma 12 23
```

program powinien wyświetlić na ekranie liczbę `35` (`12 + 23 = 35`). Zmodyfikować program tak, by obliczał sumę dowolnej liczby liczb. Funkcje `atoi()` oraz `atol()` pozwalają zamienić tekst (będący zapisem liczby całkowitej) na liczbę całkowitą (reprezentację binarną).

Zadanie 9 Napisz własną implementację funkcji `atoi()`. Funkcja powinna działać identycznie jak ta z biblioteki standardowej. Przetestuj funkcję w prostym programie.

Zadanie 10 Napisz program, który tworzy plik (`data.txt`) zwierając 1000 wierszy. Każdy wiersz powinien zawiercić następująca dane: pierwszym elementem powinna być jedna z trzech liczb `0`, `1` lub `2` natomiast po spacji powinien być umieszczony losowy znak drukowalny, gdy pierwszym elementem wiersza jest `0`; losowa liczba całkowita z przedziału od 0 do 5000, gdy pierwszym elementem wiersza jest `1`; losowa liczba zmiennoprzecinkowa z przedziału od 0 do 100, gdy pierwszym elementem wiersza jest `2`. Same wiersze również powinny być generowane losowo. Przykładowo, pierwsze pięć wierszy mogłoby wyglądać następująco:

```
1 568
```

```
0 &
1 1456
2 45.345
0 A
```

Napisz program stosując paradygmat proceduralny tj. zaprojektuj odpowiednie funkcje wykonujące poszczególne zadania.

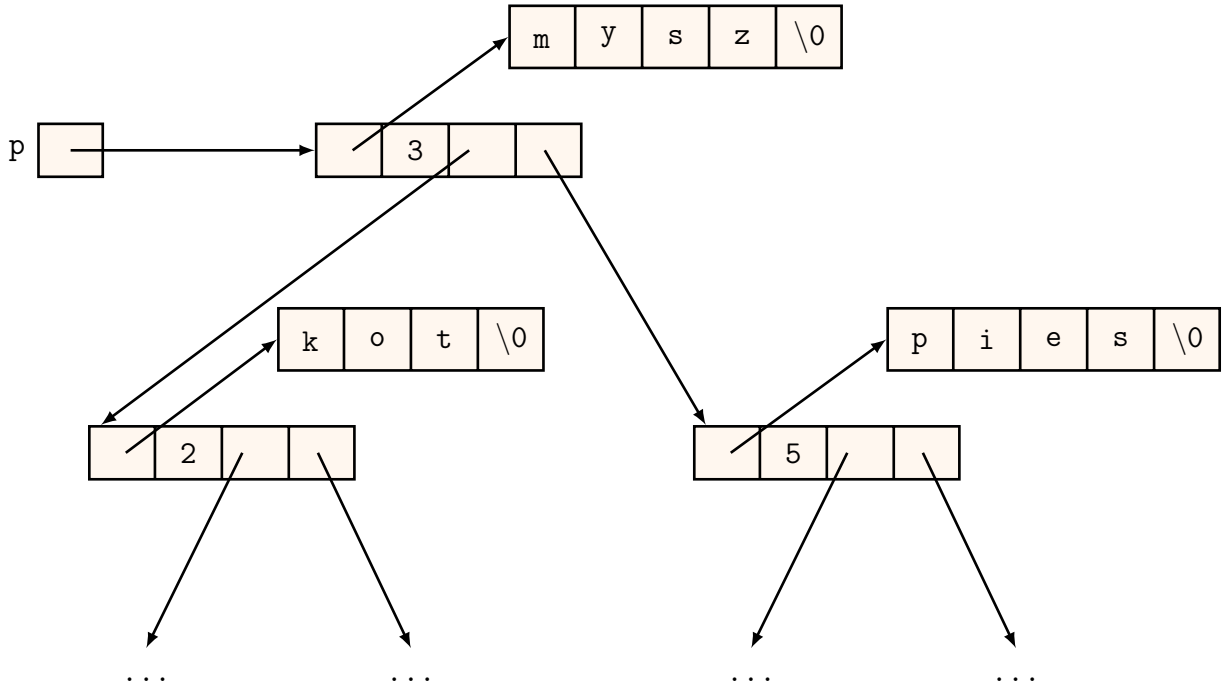
Zadanie 11 Napisz program, który odczytuje dane z pliku (`data.txt`) utworzonym w zadaniu 10 i zapisuje je do tablicy struktur typu `data` - tablica powinna być alokowana dynamicznie. Struktura `data` powinna zawierać dwa elementy: tzw. wyróżnik pola unii (stała wyliczeniowa zdefiniowana za pomocą typu wyliczeniowego (`enum`) np. `CHARACTER`, `INTEGER`, `FLOATING`), unia anonimowa o trzech polach typu `char`, `int` i `float`. Jeżeli plik `data.txt` zawiera np. wiersze:

```
1 122
0 x
2 4.75
```

To pierwszych elementem tablicy powinna być struktura o wyróżniku równym `INTEGER` i unii przechowującej liczbę całkowitą `122`; drugim elementem tablicy powinna być struktura o wyróżniku równym `CHARACTER` i unii przechowującej wartość `'x'`; trzecim elementem tablicy ma być struktura o wyróżniku pola unii równym `FLOATING` i unii przechowującej wartość `4.75`. Program powinien wyświetlać uzupełnioną tablicę danych. Napisz program stosując paradygmat proceduralny tj. zaprojektuj odpowiednie funkcje wykonujące poszczególne zadania.

Zadanie 12 Uzupełnij program z zadania 11 o funkcję sortująca tablicę struktur. Tablica powinna być posortowana w taki sposób, że najpierw znajdują się więcej struktury zawierające znaki, później liczby całkowite, a na końcu liczby zmiennoprzecinkowe. Każda z trzech grup również powinna być posortowana w porządku niemalejącym (znaki powinny być sortowane leksykograficznie). Wykorzystaj znaną Ci technikę sortowania np. sortowanie bąbelkowe, sortowanie przez wybieranie, sortowanie przez wstawianie lub inną.

Zadanie 13 Napisz program, który będzie odczytywał kolejne wyrazy z pliku tekstowego i umieszczał je w drzewie BST obliczając w przypadku wyrazów występujących wielokrotnie liczbę wystąpień (zob. rysunek 4). Podczas tworzenia drzewa zostanie wykorzystany porządek alfabetyczny wyrazów. Po "lewej stronie" będą umieszczone wyrazy wcześniejsze według tego porządku, po "prawej stronie" późniejsze. Drzewo będzie zbudowane ze struktur zawierających cztery składowe: wskaźnik wskazujący pierwszy element tablicy zawierającej wyraz, składową całkowitą podającą liczbę wystąpień wyrazu, dwa wskaźniki wskazujące lewego i prawego potomka. Jeśli węzeł nie posiada lewego lub prawego potomka, to odpowiedni wskaźnik będzie miał wartość `NULL`.



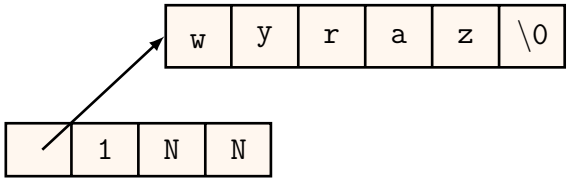
Rys. 4: Przykładowe drzewo BST.

i) Napisać definicję opisanej powyżej struktury.

ii) Napisać funkcję tworzącą nową strukturę. Funkcja będzie wykorzystywana, gdy z pliku zostanie odczytany wyraz, którego nie ma jeszcze w drzewie (bo został odczytany z pliku po raz pierwszy). Argumentem funkcji ma być tablica zawierająca odczytany wyraz. Funkcja powinna przydzielić, korzystając z funkcji `malloc()`, pamięć dla struktury oraz dla tablicy przeznaczonej do przechowywania wyrazu. Następnie należy skopiować do tej tablicy wyraz, który został przekazany jako argument funkcji (przydatne mogą być funkcje `strlen()` oraz `strcpy()`). Składowej całkowitej struktury (podającej liczbę wystąpień wyrazu) należy przypisać wartość `1`, a wskaźnikom wskazującym potomków wartość `NULL`. Funkcja powinna zwrócić wskaźnik wskazujący utworzoną strukturę. Na poniższym rysunku `N` oznacza wskaźnik pusty (`NULL`);

iii) Napisać funkcję "dodającą do drzewa jeden wyraz". Tablica zawierająca ten wyraz ma być argumentem funkcji. Przyjmujemy, że struktura reprezentująca korzeń drzewa jest wskazywana przez wskaźnik będący zmienną zewnętrzną (globalną). Na rysunku jest to wskaźnik `p`. Jeśli drzewo jest puste, to wskaźnik ten przyjmuje wartość NULL. W tym przypadku funkcja powinna przypisać temu wskaźnikowi adres struktury utworzonej za pomocą funkcji opisanej w punkcie (ii). Jeśli drzewo jest niepuste, to funkcja powinna sprawdzić, czy "dodawany wyraz" występuje już w drzewie. Jeśli tak jest, to należy zwiększyć o jeden składową podającą liczbę wystąpień wyrazu w pliku. Jeśli w drzewie

nie ma struktury z tym wyrazem, to należy ją utworzyć za pomocą funkcji z punktu (ii) i umieścić jako liść (tzn. węzeł, który nie ma potomka) w odpowiednim miejscu drzewa. wskazujący utworzoną strukturę. Na poniższym rysunku `N` oznacza wskaźnik pusty (`NULL`);



Wskazówka: Poszukując struktury zawierającej wyraz będący argumentem funkcji, posłużyć się dwoma wskaźnikami. Jeden ze wskaźników należy "przesuwać" do kolejnych węzłów drzewa (rozpoczynając od korzenia i "przechodzić" do prawego lub lewego potomka. Kierunek, w którym należy "przesuwać" wskaźnik wynika ze sposobu uporządkowania wyrazów umieszczonych w drzewie ("po lewej stronie" wcześniejsze w porządku alfabetycznym, "po prawej" późniejsze). Do porównywania wyrazów można użyć funkcji `strcmp()`. Przed "przesunięciem" wskaźnika należy skopiować jego wartość do drugiego wskaźnika; Jeśli w drzewie nie ma poszukiwanego wyrazu, to gdy pierwszy wskaźnik przyjmie wartość NULL, drugi wskaże strukturę, do której należy "doczepić" strukturę zawierającą wstawiany wyraz.

iv) Napisać funkcję wyszukującą w drzewie wyraz podany, jako jej argument. Funkcja powinna zwrócić wskaźnik wskazujący strukturę zawierającą poszukiwany wyraz lub wskaźnik pusty, jeśli wyraz ten w drzewie nie występuje.

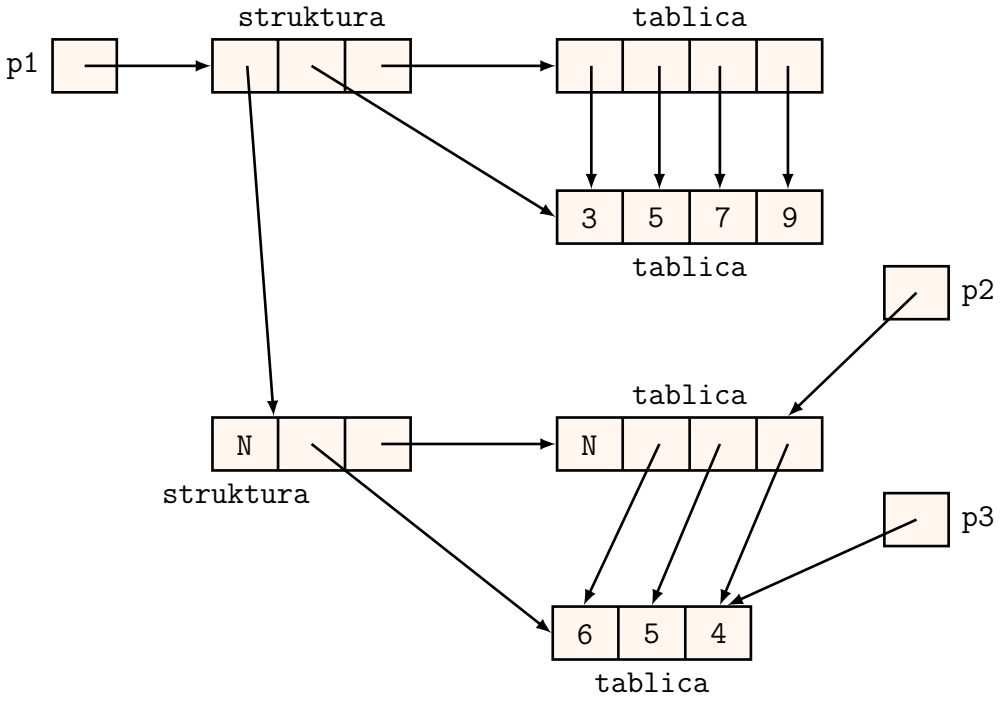
v) Napisać funkcję rekurencyjnie wyświetlającą w porządku alfabetycznym wyrazy z liczbą ich wystąpień. Argumentem funkcji powinien być wskaźnik wskazujący strukturę reprezentującą węzeł drzewa.

vi) Napisz funkcję `czytaj_wyraz()` odczytującą jeden wyraz z pliku i umieszczającą go w tablicy przekazanej za pomocą argumentu. Funkcja powinna zamieniać odczytywane duże litery na małe (by nie liczyć osobno wyrazów zawierających duże litery, można wykorzystać funkcję `tolower`).

vii) Napisać program podający wyrazy występujące we wskazanym pliku wraz z liczbą wystąpień. Po odczytaniu wszystkich wyrazów, program powinien zapisać je w innym pliku w porządku alfabetycznym.

Zadanie 14 Napisać program, w którym przydzielana jest dynamicznie pamięć dla przedstawionych na rysunku 5 tablic i struktur. Program powinien przypisać pokazane na rysunku

wartości wskaźnikom `p1`, `p2` i `p3` oraz tablicom i strukturom (`N` oznacza wskaźnik pusty `NULL`, każdy ze wskaźników albo wskazuje element tablicy albo strukturę (całą)). Dobrać odpowiednio typy, tak by podczas kompilacji programu nie było komunikatów o błędach. W programie można użyć dodatkowe wskaźniki (poza przedstawionymi na rysunku). Nie używać operatorów rzutowania (konwersji typu).



Rys. 5: Przykładowa struktura danych.

Zadanie 15 Napisz program, który realizuje menu za pomocą tablicy wskaźników do funkcji. Na przykład wybranie opcji `'a'` powodowałoby uruchomienie funkcji wskazywanej przez pierwszy element tablicy.

Zadanie 16 Napisz funkcję `transform()`, która pobiera cztery argumenty, nazwę tablicy źródłowej, zawierającej dany typu `double`, nazwę docelowej tablicy typu `double`, zmienną typy `int` określającą liczbę elementów tablicy oraz nazwę funkcji (albo odpowiedni wskaźnik do funkcji). Funkcja `transform()` powinna zastopować podana funkcję będąca jej argumentem do każdego elementu tablicy źródłowej, umieszczając rezultaty w docelowej tablicy. Na przykład wywołanie:

```
transform(source, target, 100, sin)
```

powinno wstawić do `target[0]` wartość `sin(source[0])` itd. Przetestuj działanie funkcji `transform()` w programie stosując ją czterokrotnie - dla dwóch funkcji z biblioteki `math.h` oraz dwóch funkcji autorskich.