Wykład 6_1
Abstrakcyjne typy danych – stos
Realizacja tablicowa i za pomocą rekurencyjnych typów danych

- **Abstrakcyjny typ danych**

  Klient korzystający z abstrakcyjnego typu danych:
  - ma do dyspozycji **jedynie operacje** typu wstaw(), usun(), wyszukaj() itp.,
  - natomiast **nie ma wglądu na typy danych**, które w rzeczywistości zastosowano do przechowywania danych
  - nawet, jeśli zmieni się implementacja abstrakcyjnego typu danych a nagłówki funkcji nie zostaną zmienione, programy korzystające z tego typu pozostają bez zmian.

2. Stos – algorytm

Stos jest jednym z najważniejszych typów danych, które wstawiają i usuwają dane ze zbioru danych

Przykłady zastosowań stosu: obliczanie wyrażenia postfiksowego, wyrażenia infiksowe (dwa stosy), przekształcenia wyrażenia infiksowego w postfiksowe itp

Etap 1 - Opis ADT

Nazwa typu: Stos elementów
Własności typu: Potrafi przechować ciąg elementów o dowolnym rozmiarze
Dostępne działania:
Inicjalizacja stosu
Określenie, czy stos jest pusty
Dodanie elementu do stosu,
Usuwanie ze stosu,

Etap 2 - Budowa interfejsu

```
typedef  int  dane;                    {typ informacji umieszczanej na stosie}

typedef struct ELEMENT* stos;          {typ wskazania na element stosu }
struct ELEMENT                         {typ  elementu stosu }
  {dane Dane;
   stos Nastepny;
  };
```

void Inicjalizacja(stos & Stos);
{*działanie:* inicjuje stos
warunki wstępne: Stos wskazuje na pierwszy element
warunki końcowe: stos zostaje zainicjowany jako pusty}

inline int Pusty(stos Stos) { **return** Stos==NULL;}
{*działanie*: określa, czy stos jest pusty; typ **inline**, bo często wywoływana
warunki wstępne: *Stos* jest zainicjowany,
warunki końcowe: funkcja zwraca 1, jeśli stos pusty, w przeciwnym wypadku 0}

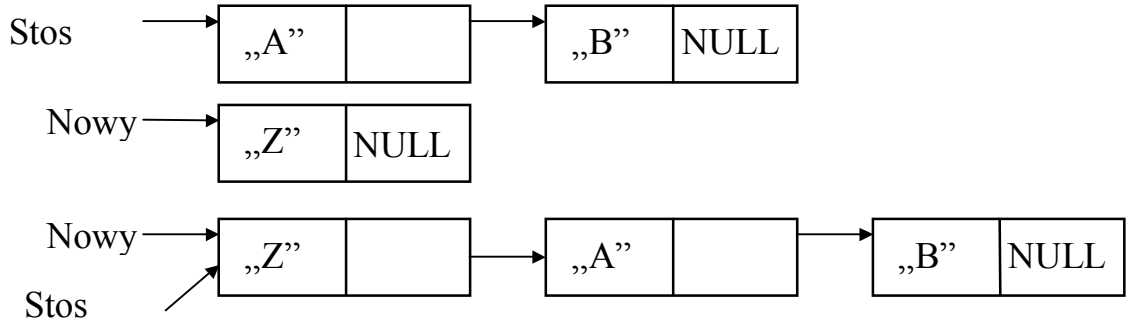
int Wstaw(stos& Stos, dane Dana);
{*działanie:* dodaje element na początek ciągu, zwany szczytem stosu *Stos*
warunki początkowe: Dana jest daną do wstawienia na szczyt zainicjowanego stosu
warunki końcowe: jeśli to możliwe, funkcja dodaje daną *Dana* na szczyt stosu i zwraca wartość 1, w przeciwnym wypadku 0 }

dane Usun(stos& Stos);
{*działanie:* jeśli stos nie jest pusty, usuwa element ze szczytu stosu, czyli element ostatnio wstawiony do stosu
warunki początkowe: Stos jest zainicjowanym niepustym stosem
warunki końcowe: funkcja usuwa element na szczycie stosu i zwraca umieszczoną tam daną}

Etap 3. Implementacja stosu za pomocą rekurencyjnej struktury danych

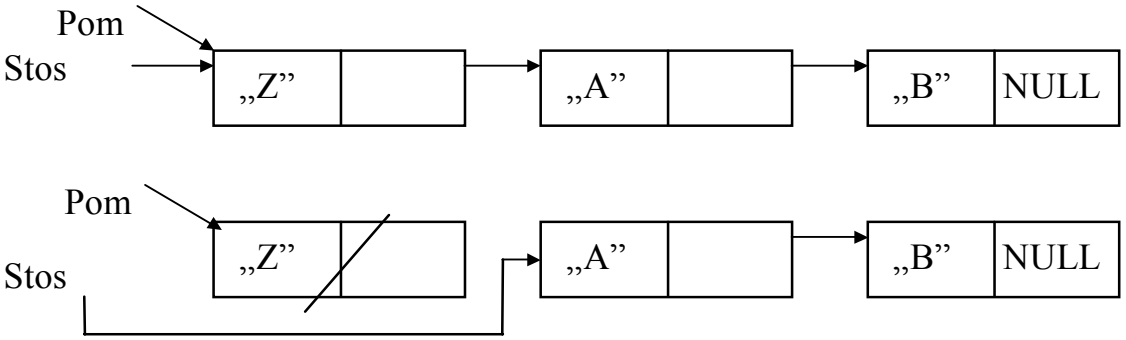
- **void** Inicjalizacja(stos& Stos) { Stos = NULL; }

- wstawianie elementów zawsze na początek struktury




```
int Wstaw(stos& Stos, dane Dana)
 { stos Nowy;
   Nowy = new ELEMENT;
   if (!Pusty(Nowy))
     Nowy->Dane=Dana;
   else return 0;
   Nowy->Nastepny= Stos;
   Stos= Nowy;
   return 1;
  }
```

- usuwanie elementów zawsze na początku struktury




```
dane Usun(stos& Stos)
  {stos Pom;
   dane d;
   Pom = Stos;                    //zapamiętanie pierwszego elementu do usunięcia
   Stos = Stos->Nastepny;         //odłączenie pierwszego elementu od listy
   d= Pom->Dane;                  //((*Pom).Dane)
   delete Pom;                    //usunięcie pierwszego elementu z pamięci}
   return d;
  }
```

Obliczanie wyrażeń postfiksowych za pomocą stosu

Wyrażenie infiksowe (((1+2)*(3*4))+(5*2))

Wyrażenie postfiksowe 1 2 + 3 4 **5 2 *+

Obliczanie wyrażenia postfiksowego

1 2 + 3 4 **5 2 *+
(1+ 2) 3 4 **5 2 *+
3 (3*4)* 5 2 *+
(3* 12) 5 2*+
36 (5 * 2) +
36 + 10
46

Algorytm obliczania wyrażenia postfiksowego

- Ustaw i=0
- Wykonuj, co następuje, aż zostaną wyczerpane znaki wyrażenia postfiksowego:

  2.1. Jeśli znakiem wyrażenia jest operator, to **Usun** ze stosu dwie kolejne dane, wykonaj działanie zgodne z operatorem i **Wstaw** wynik na stos
  - Jeśli znakiem jest cyfra, **Wstaw** na stos wartość 0
  - Dopóki i-tym znakiem jest cyfra, wykonuj //oblicza wartość argumentu
    - **Usun** ze stosu daną i pomnóż przez 10 oraz dodaj do tego wyniku wartość liczby jako jednocyfrowej wartości
    - **Wstaw** tę wartość na stos
    - Zwiększ licznik znaków wyrażenia o 1, i++

  2.4. Zwiększ licznik znaków wyrażenia o 1, i++ //opuszcza spację

Operator	Obliczenia	Stos			
		1	2	3	4
		1	2		
+	1+2=3				
		4	3	3	
*	4*3=12	3			
		12	3		
*	12*3=36				
		36			
		2	5	36	
*	2*5=10	36			
		10	36		
+	10+36=46				
		46			

Przykład programu

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

      //1. interfejs ADT stosu
typedef int dane;                          // dane umieszczone stosie
typedef struct ELEMENT* stos;              //nazwa wskaźnika na element stosu
struct ELEMENT
 {
  dane Dane;
  stos Nastepny;
 };
                  //funkcje ADT stosu
void Inicjalizacja(stos& Stos);
inline int Pusty(stos Stos);
int Wstaw(stos& Stos, dane Dana);
dane Usun(stos& Stos);

      //2. funkcje we/wy dla danych umieszczonych na stosie
void Pokaz_dane (dane Dana);          //funkcja do wyświetlania danej
dane Dane();                          // funkcja do wprowadzania danej

      //3. funkcje ogolnego przeznaczenia
 void Komunikat(char*);                        //funkcja do wyświetlania komunikatów
 char Menu(const int ile, char *Polecenia[]);  //funkcja do wyświetlania tablicy
                  //łańcuchów zawierających menu programu i zwracająca naciśniety klawisz

      //4. elementy programu
const int Esc=27;
const int POZ=5;
char * Tab_menu[POZ] = { "1 : Wstawianie do stosu- na poczatek",
                         "2 : Usuwanie ze stosu-na poczatku",
                         "3 : Wydruk stosu wraz z jego usuwaniem",
                         "4 : Obliczanie wyrazenia postfiksowego",
                        " >Esc Koniec programu"};

                  //funkcja klienta korzystające ze stosu
void Wyswietl_usun_ze_stosu(stos& Stos); //wyświetlanie stosu
dane Oblicz(char* w, stos& Stos); //zastosowanie stosu do obliczenia wyrażenia
                                  //postfiksowego
void main(void)
```

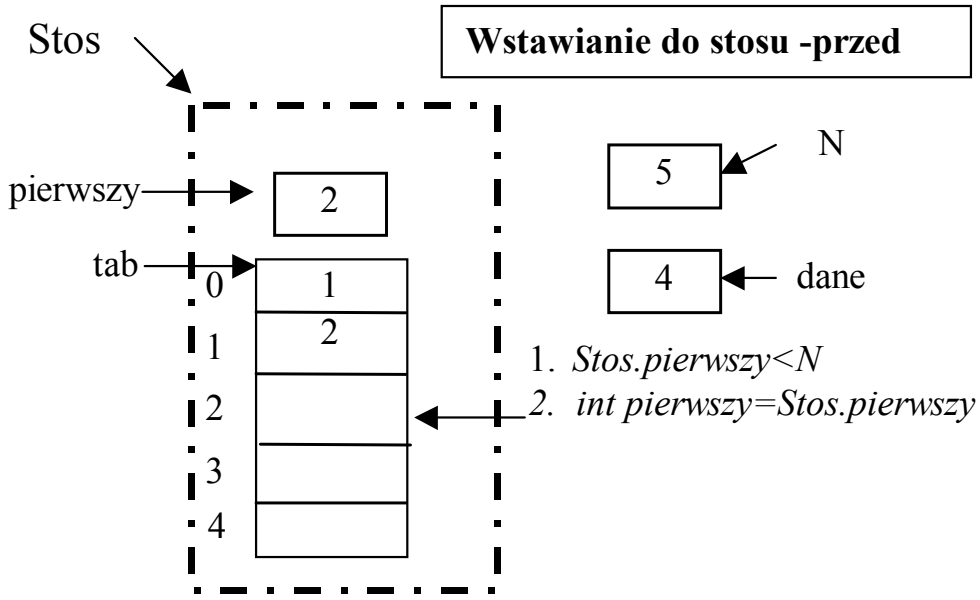
```
{ stos Stos;
  char Wybor;
  char w[19]= {'5',' ','9','1',' ','8','2',' ','+','4',' ','6',' ','*','*','7',' ','+','*'};
  dane Dana;
  clrscr();
  Inicjalizacja(Stos);
  do
     { Wybor= Menu(POZ, Tab_menu);
       switch (Wybor)
        {
          case '1' :  Dana=Dane();
                      if (Wstaw(Stos, Dana)==0)
                        Komunikat("Brak pamieci\n");
                      break;
          case '2' :  if (Pusty(Stos))
                         Komunikat("\nStos pusty\n");
                      else (Usun(Stos));
                      break;
          case '3' :  if (Pusty(Stos))
                        Komunikat("\nStos pusty\n") ;
                      else Wyswietl_usun_ze_stosu(Stos);
                    break;
          case '4' :  printf("\n%d\n", Oblicz(w, Stos));
                      getch();
        }
     } while (Wybor !=Esc );
 }

//**************funkcje klienta korzystające ze stosu***********

void Wyswietl_usun_ze_stosu(stos& Stos)
   {dane d;
    while (!Pusty(Stos))
     {
      d=Usun(Stos);
      Pokaz_dane(d);
     }
   }
```

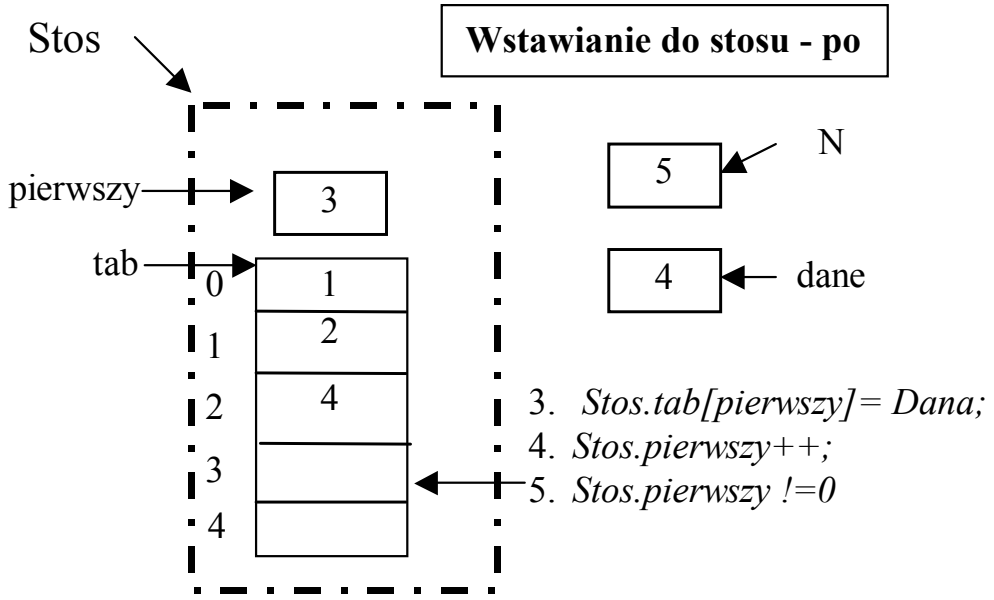
```
dane Oblicz(char* w, stos& Stos) //wg Robert Sedgewick „Algorytmy w C++”
 {
   for ( int i=0;i<19; i++)
    {  if (w[i]=='+')
          Wstaw(Stos,Usun(Stos)+Usun(Stos));
       if (w[i]=='*')
          Wstaw(Stos,Usun(Stos)*Usun(Stos));
       if (w[i]>='0' && w[i]<='9')
         Wstaw(Stos,0);
       while(w[i]>='0' && w[i]<='9')
          Wstaw(Stos,10*Usun(Stos)+(w[i++]-'0'));
     }
    return Usun(Stos);
 }

//*********funkcje interfejsu ADT stosu************

 void Inicjalizacja(stos& Stos)
  {  Stos = NULL; }

 inline int Pusty(stos Stos)
  { return Stos==NULL;   }

 int Wstaw(stos& Stos, dane Dana)
 { stos Nowy;
   Nowy = new ELEMENT;
   if (!Pusty(Nowy))   Nowy->Dane=Dana;
   else return 0;

   Nowy->Nastepny= Stos;
   Stos= Nowy;
   return 1;
  }

 dane Usun(stos& Stos)
  {stos Pom;
   dane d;
   Pom = Stos;
   Stos = Stos->Nastepny;
   d= Pom->Dane;                              //((*Pom).Dane)
   delete Pom;
   return d; }
```

```
//*********funkcje ogólnego przeznaczenia************

char Menu(const int ile, char *Polecenia[])
  {
   clrscr();
   for (int i=0; i<ile;i++)
        printf("\n%s",Polecenia[i]);
   return getch();
  }

 void Komunikat(char* s)
  {
   printf(s);
   getch();
  }

//*******funkcje we/wy dla danych umieszczonych na stosie*********

dane Dane()
   { int a;
    do
     { fflush(stdin);
       printf("\n\nPodaj dane typu int: ");
     } while (scanf("%d",&a)!=1);
    return a;
   }

  void Pokaz_dane(dane Dana)
   {
    printf("\nNumer: %d\n", Dana);
    printf("Nacisnij dowolny klawisz...\n");
    getch();
   }
```

Etap 3. Implementacja stosu za pomocą tablicy

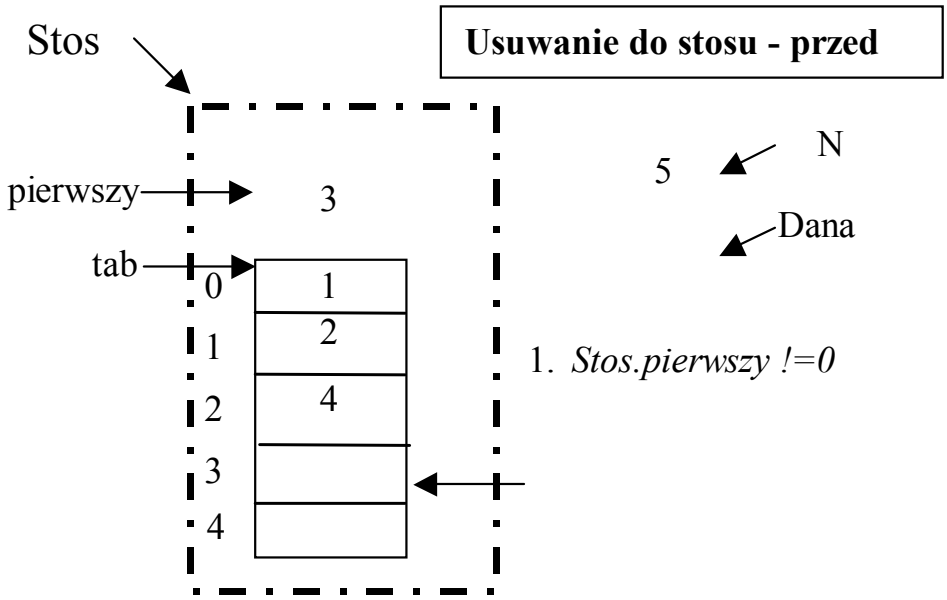
- wstawianie

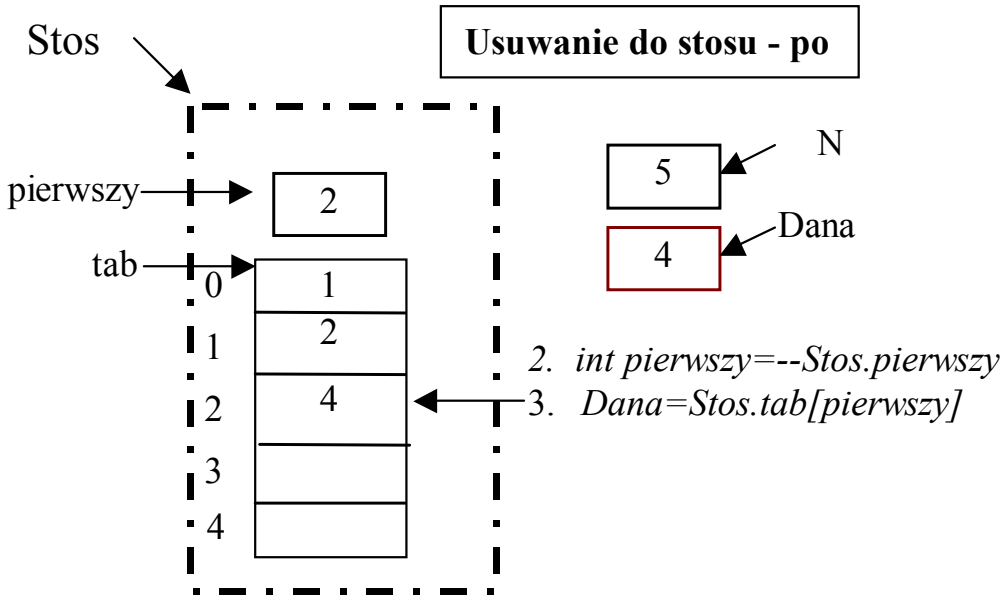





```
int Wstaw(stos& Stos, dane Dana)
 { if (Stos.pierwszy==N) return 0;  //nie można wstawić do stosu, gdy jest pełen
   int pierwszy=Stos.pierwszy++;
   Stos.tab[pierwszy]= Dana;
   return 1;
  }
```

- Usuwanie ze stosu






```
dane Usun(stos& Stos)
  {
   dane d;
   int pierwszy= --Stos.pierwszy;   //wyznaczenie indeksu elementu usuwanego ze stosu
   d= Stos.tab[pierwszy];
   return d;
  }
```

Implementacja stosu za pomocą tablicy

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

```
    //1. interfejs ADT stosu
typedef int dane;                          // dane umieszczone stosie
const long N=10;                           //rozmiar stosu
struct stos                                //typ struktury stosu
 { int pierwszy;                           //liczba elementów
   dane tab[N];                            //tablica danych
 };
                    // interfejs stosu się nie zmienił, definicja zmieniła się!!!
void Inicjalizacja(stos& Stos);
inline int Pusty(stos Stos);
int Wstaw(stos& Stos, dane Dana);
dane Usun(stos& Stos);

                    // Te funkcje się nie zmieniły – ani nagłówki, ani definicja!!!

    //2. funkcje we/wy dla danych umieszczonych na stosie

void Pokaz_dane (dane Dana);
dane Dane();

   //3. funkcje ogolnego przeznaczenia

 void Komunikat(char*);
 char Menu(const int ile, char *Polecenia[]);

   //4. elementy programu
const int Esc=27;
const int POZ=5;
char * Tab_menu[POZ] = {"1 : Wstawianie do stosu- na poczatek",
                        "2 : Usuwanie ze stosu-na poczatku",
                        "3 : Wydruk stosu wraz z jego usuwaniem",
                        "4 : Obliczanie wyrazenia postfiksowego",
                        " >Esc Koniec programu"};

          //funkcje klienta korzystajace ze stosu

void Wyswietl_usun_ze_stosu(stos& Stos);
dane Oblicz(char* w, stos& Stos);
```

```
void main(void)                          //tekst funkcji main się nie zmienił
 { stos Stos;
   char Wybor;
   char w[19]={'5',' ','9','1',' ','8','2',' ','+','4',' ','6',' ','*','*','7',' ','+','*'};
   dane Dana;
   clrscr();
   Inicjalizacja(Stos);
   do
       { Wybor= Menu(POZ, Tab_menu);
        switch (Wybor)
           {
           case '1' : Dana= Dane();
                      if (Wstaw(Stos, Dana)==0)   Komunikat("Brak pamieci\n");
                      break;
           case '2' : if (Pusty(Stos))  Komunikat("\nStos pusty\n");
                      else (Usun(Stos));
                      break;
           case '3' : if (Pusty(Stos))  Komunikat("\nStos pusty\n") ;
                      else Wyswietl_usun_ze_stosu(Stos);
                      break;
           case '4' : printf("\n%d %d\n",Oblicz(w, Stos));  getch();
    }
   } while (Wybor !=Esc );
  }

//*********funkcje interfejsu ADT stosu************

void Inicjalizacja(stos& Stos)
 {  Stos.pierwszy = 0; }

inline int Pusty(stos Stos)
  {  return Stos.pierwszy==0; }   //stos jest pusty, gdy liczba elementów jest równa zero

int Wstaw(stos& Stos, dane Dana)
  { if (Stos.pierwszy==N) return 0;  //nie można wstawić do stosu, gdy jest pełen
    int pierwszy=Stos.pierwszy++;
    Stos.tab[pierwszy]= Dana;
    return 1;
   }

dane Usun(stos& Stos)
   {dane d;
    int pierwszy= --Stos.pierwszy;   //wyznaczenie indeksu elementu usuwanego ze stosu
    d= Stos.tab[pierwszy];
    return d;
   }
```

Podsumowanie

1. Pojęcie rekurencyjnych typów danych
2. Klasyfikacja struktur danych
3. Pojęcie abstrakcyjnego typu danych

1. Pojęcie rekurencyjnych typów danych

Przykład (*Wirth N. „Algorytmy + Struktury Danych = Programy", WNT 1989*)

a) deklaracja modelu drzewa dziedziczenia:

```
type osoba =   record
                 if znany then
                 ( imię : alfa;
                   ojciec, matka: osoba)
                 end;
```

b) opis reprezentacji zmiennej *rodowód* typu *osoba*:

jeśli *znany = False*, to istnieje tylko pole znacznikowe *znany* równe *False* (F)

jeśli *znany = True (T),* to istnieją jeszcze trzy pola (*imię, ojciec, matka*)

c) przypadek danych: *rodowód* =

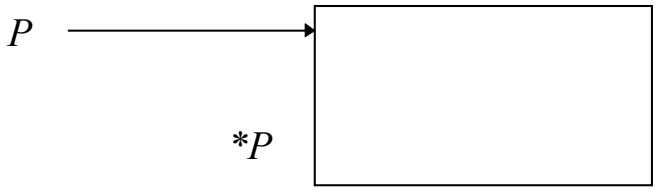
```
(True, Jan,
      (True, Marek,
            (True, Adam,
               (False),
               (False)
             ),
             (False)
       ),
      (True, Maria,
            (False),
            (True, Ewa,
                 (False),
                 (False)
             )
      )
 )
```

2. Klasyfikacja struktur danych

- Struktury danych można podzielić na:

1) typy wymagające deklarowania rozmiarów - realizowane jako tablice lub struktury z bezpośrednim dostępem do każdego elementu tych struktur za pomocą operatorów indeksowania „[ ]” lub wyboru: „->” oraz „.”
2) typy nie wymagające deklarowania rozmiarów - rekurencyjne typy danych realizowane jako dynamiczne struktury danych z pośrednim dostępem do ich elementów. Rekurencyjny typ danych:
   2.1) zawiera elementy jako struktury
   2.2) używa wskaźników do wskazania następnego elementu typu,
   2.3) używa dynamicznego przydziału pamięci dla swoich elementów
   2.4) algorytm dostępu do poszczególnych składowych tej struktury określa programista dzięki jawnemu użyciu wskaźników.
2) uporządkowane: jako kryterium wstawiania danych do struktury jest ich wartość: drzewa poszukiwań binarnych, listy posortowane
3) nieuporządkowane: kryterium wstawiania danych do struktury dotyczy priorytetu danych: stosy, kolejki, listy nieposortowane, drzewa, grafy

- Dynamiczne przydzielanie pamięci zmiennej *typ* p=**new** typ.*



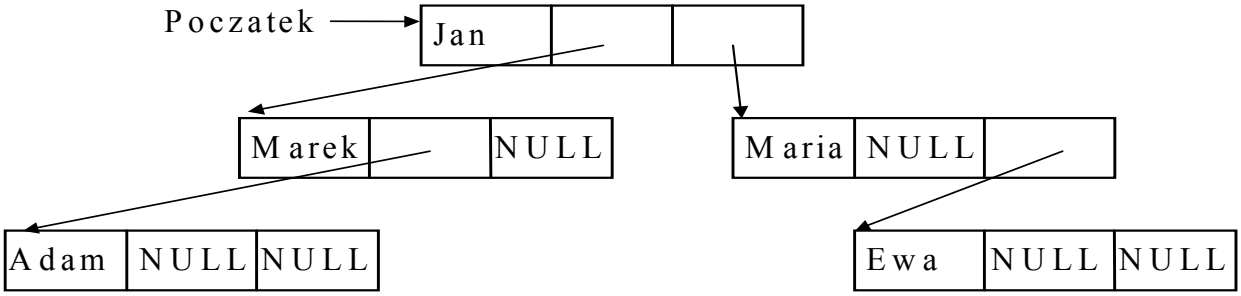
- Wskaźnikowy model rekurencyjnych typów danych:

Przykład Wskaźnikowy model rodowodu

a) deklaracja typu

```
struct osoba
 { char imie [10];
   osoba* ojciec, *matka;
 };
```

- wskaźnikowa struktura rodowodu `osoba* Poczatek;`



2. Abstrakcyjne typy danych

3.1.Główne cechy typów danych stosowanych w programach:

1) muszą być dostosowane do rozwiązywanego problemu
2) muszą zawierać dwa rodzaje informacji:
   2.1) zbiór własności
   2.2) zbiór działań.

Np. typ **int**
własności: reprezentuje liczby całkowite o wartościach od -32768 do +32767 zakodowanych w kodzie dwójkowym na dwóch bajtach
działania: zmiana znaku, dodawanie, mnożenie, dzielenie, modulo...

3.2. Trzy etapy procesu definiowania typów przez programistę [wg Stephen Prata - Szkoła programowania, Język C]:

- Przygotowanie opisu ADT (abstrakcyjnego typu danych).

Abstrakcyjny opis obejmuje własności typu i operacji, jakie można na nim wykonać. Opis ten nie powinien być związany z żadną konkretną implementacją i językiem programowania. Ta formalna charakterystyka nosi nazwę abstrakcyjnego typu danych ADT.

b) Opracowanie interfejsu programistycznego realizującego ADT -

Jest to wskazanie sposobu przechowywania danych i opisanie zbioru funkcji wykonujących potrzebne operacje. W języku C/C++ etap ten może polegać na utworzeniu definicji struktury i prototypów funkcji przetwarzających. Funkcje te pełnią dla nowego typu tę samą rolę, co operatory w przypadku podstawowych typów języka C/C++. Utworzony interfejs będzie stosowany przez osoby, które chcą skorzystać z nowego typu danych

- Pisanie kodu implementującego interfejs

Ten krok jest kluczowy, w którym należy szczegółowo zrealizować wymagania wynikające z opisu. Należy zauważyć, że programista korzystający z interfejsu nie musi już orientować się w szczegółach implementacji

3.3 Podsumowanie:

Programista korzystający z abstrakcyjnego typu danych:

- może **wielokrotnie używać** utworzony typ danych w programach
- ma do dyspozycji **jedynie operacje** typu wstaw(...), usun(...), wyszukaj(...)
- natomiast **nie ma wglądu na typy danych**, które w rzeczywistości zastosowano do przechowywania danych
- dlatego nawet, jeśli zmieni się implementacja abstrakcyjnego typu danych a nagłówki funkcji nie zostaną zmienione, **programy źródłowe** korzystające z tego typu **pozostają bez zmian**.