Wykład 7

Abstrakcyjne typy danych – słownik (lista symboli)

Definicja słownika:

Słownik (tablica lub lista symboli) to struktura danych zawierająca elementy z kluczami, która pozwala na przeprowadzanie dwóch podstawowych operacji: wstawiania nowego elementu i zwrot elementu z podanym kluczem (R.Sedgewick,Algorytmy w C++)

Zastosowania słownika (wg R.Sedgewick,Algorytmy w C++)

- Organizowanie informacji
- Organizacja oprogramowania systemów komputerowych (dla programów listy symboli występują w roli słowników: klucze są nazwami symbolicznymi występującymi w programach, elementy opisują nazywany przedmiot)
- Organizacja sieci komputerowych, gdzie stosuje się nazwy symboliczne
- pamięć skojarzeniowa na poziomie sprzętu

Implementacje słowników (listy symboli):

- ✓ Tablice indeksowane kluczem (dla niewielkiej liczby kluczy – wykład 7)
- ✓ Tablice uporządkowane – (ćwiczenie 7)
- ✓ Listy wiązane uporządkowane – (ćwiczenie 7)
- ✓ Drzewa poszukiwań binarnych – (Algorytmy i struktury danych 2)
- ✓ .....

Słownik (lista symboli)

Etap 1 - Opis ADT

Nazwa typu: Lista elementów z kluczami (symbolami)

Własności typu: Potrafi przechować ciąg elementów i zwraca element z podanym kluczem

Dostępne działania: Inicjalizacja słownika
Wyszukanie jednego lub wielu elementów z kluczem
Dodanie nowego elementu
Usuwanie określonego elementu
Wybranie k-tego największego elementu
Pokazanie wszystkich elementów w kolejności kluczy
Podanie liczby kluczy

Etap 2 - Budowa interfejsu

```
const int Maks=10;
const int c_zero=-1;
 struct Osoba
  { int klucz;
  };
typedef  Osoba dane;
const dane zero={c_zero};
inline int f_zero(dane d, const int zero);

struct lista_s
  {   dane zero;
      int rozmiar;
      dane* tab;
  };
void Inicjalizacja(lista_s& Slownik, dane zero, int Maks);
{ działanie: inicjuje słownik
   warunki wstępne: Slownik jest pustą  listą symboli
   warunki końcowe:  Lista symboli zostaje zainicjowana jako pusta}
```

int Wstaw(lista_s& Slownik, dane Dana);
{ działanie: dodaje element do uporządkowanej listy symboli
warunki początkowe: Dana jest daną do wstawienia do zainicjowanej listy symboli Slownik
warunki końcowe: jeśli jest to możliwe, funkcja dodaje daną Dana do listy symboli Slownik i zwraca 1, w przeciwnym wypadku 0 }

int Usun(lista_s& Slownik, dane Dana);
{ działanie: usuwa element o wg klucza z listy symboli Slownik,
warunki początkowe: Slownik jest zainicjowaną niepustą listą symboli, Dana jest daną do usunięcia
warunki końcowe: usuwa element wg jego klucza z listy symboli i zwraca 1, jeśli taki element znajdował się w słowniku, w przeciwnym wypadku zwraca 2, jeśli takiego elementu nie było lub 0, jeśli podano niewłaściwy klucz (symbol)**.** Po usunięciu lista symboli może być pusta i musi być zainicjowana}

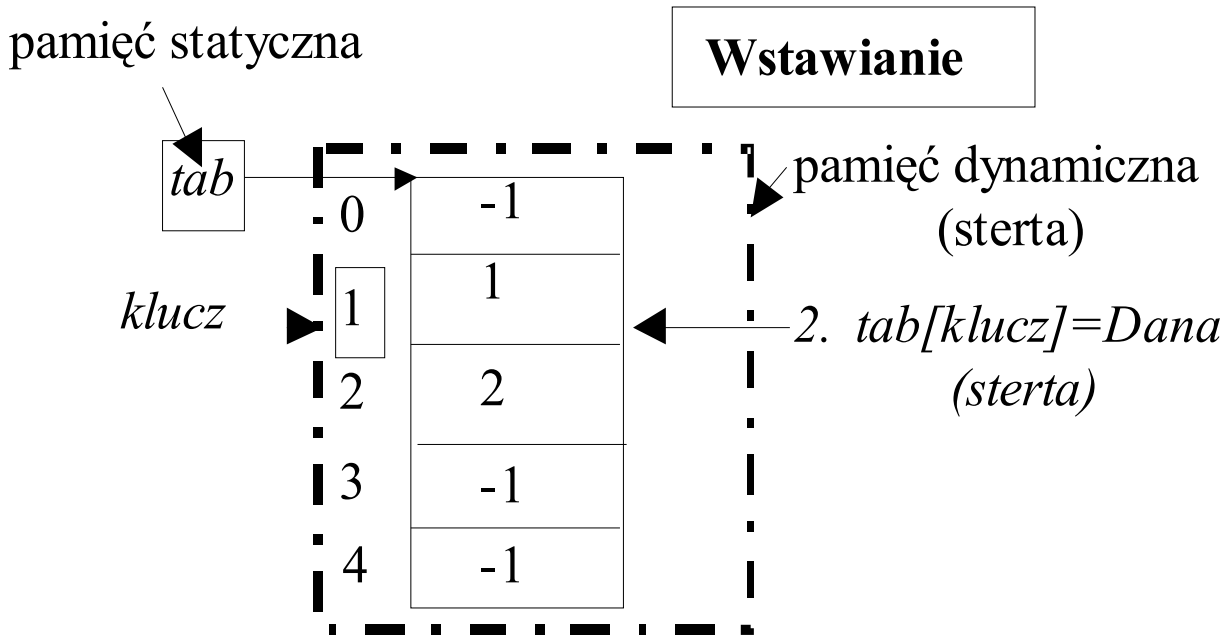
dane Szukaj_element(lista_s& Slownik, **int** Klucz);
{ działanie: podaje element wg klucza z listy symboli Slownik,
warunki początkowe: Slownik jest zainicjowaną niepustą listą symboli. Zakłada się, że klucz należy do zakresu symboli słownika
warunki końcowe: podaje przez **return** element wg jego klucza z listy symboli. }

dane Wybierz(lista_s Slownik, **int** numer);
{ działanie: podaje największy element o numerze numer z listy symboli Slownik,
warunki początkowe: Slownik jest zainicjowaną niepustą listą symboli.
warunki końcowe: podaje przez **return** element o numerze numer z listy symboli, licząc jedynie elementy z kluczem niepustym}

int Liczba_s(lista_s Slownik);
{ działanie: podaje liczbę elementów z niepustym kluczem z listy symboli Slownik,
warunki początkowe: Slownik jest zainicjowaną niepustą listą symboli.
warunki końcowe: podaje przez **return** liczbę elementów, licząc jedynie elementy z kluczem niepustym}

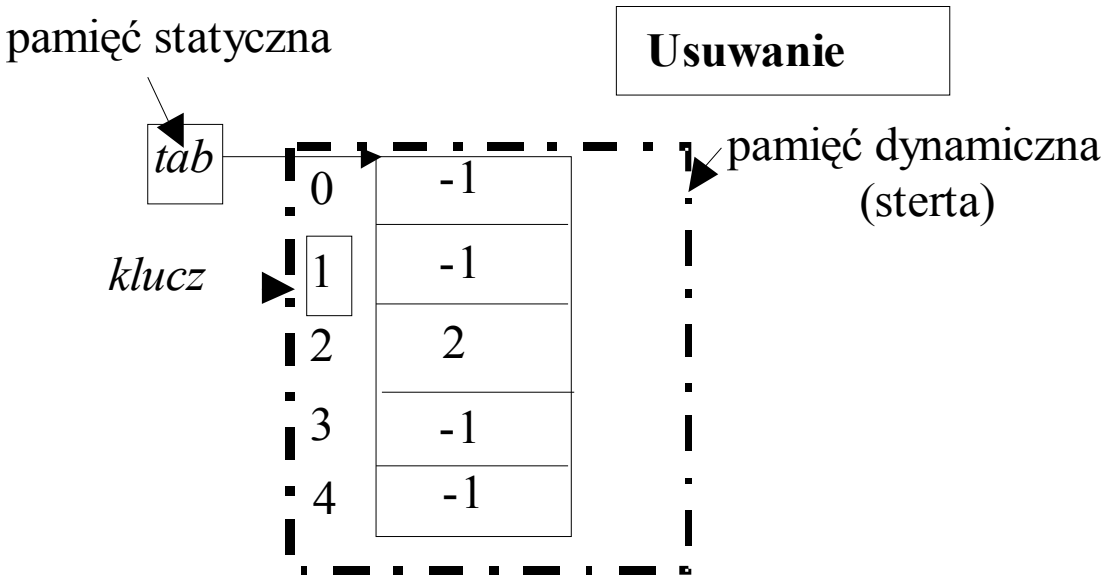
void Pokaz_liste(lista_s& Slownik, **void**(*p)(dane Dana));
{ działanie: wyświetla listę posortowaną elementów z niepustym kluczem z listy symboli Slownik,
warunki początkowe: Slownik jest zainicjowaną niepustą listą symboli.
warunki końcowe: wyświetla za pomocą funkcji wyświetlającej, podanej za pomocą wskaźnika na funkję p, jedynie elementy z kluczem niepustym}

Etap 3 - Implementacja za pomocą tablicy indeksowanej symbolami



//rozmiar=5 , klucz=Dana.klucz równy 1
*1. **if** (klucz >=0 && klucz < rozmiar)*
// ten warunek kontroluje wartość klucza

```
int Wstaw(lista_s& Slownik, dane Dana)
  {
    if  (Zakres(Slownik, Dana.klucz))
      {
        Slownik.tab[Dana.klucz]=Dana;
        return 1;
       }
    else return 0;
  }
```




```
// rozmiar=5, klucz=Dana.klucz
1. if (klucz>=0 && klucz<rozmiar)
     if (tab[klucz].klucz != -1)
      { tab[klucz].klucz= -1;
        return 1;}     //jest element o podanym kluczu
     else return 2;   //brak elementu o podanym kluczu
   else return 0;     //zły zakres klucza
```

```
int Usun(lista_s& Slownik, dane Dana)
  {
   if  (Zakres(Slownik, Dana.klucz))
      if (!f_zero(Slownik.tab[Dana.klucz], c_zero))
         {  Slownik.tab[Dana.klucz]=Slownik.zero;
            return 1;}
      else return 2;
   else return 0;
  }
```

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

        //1. interfejs ADT listy symboli
const int Maks=10;
const int c_zero=-1;
 struct Osoba
  { int klucz;
  };            // dane umieszczone liście symboli

typedef  Osoba dane;
const dane zero={c_zero};

            //funkcja obsługująca klucz elementu słownika
inline int f_zero(dane d, const int zero);

 struct lista_s
  {
    dane zero;    //dana zerowa zawierająca klucz pusty równy stałej c_zero
    int rozmiar;  //rozmiar tablicy symboli odpowiadająca zakresowi kluczy
    dane* tab; //tablica symboli
  };
            //funkcje ADT listy symboli

 void Inicjalizacja(lista_s& Lista, dane zero, int Maks);
 dane Szukaj_element(lista_s& Lista, int Klucz);
 dane Wybierz(lista_s Lista, int numer);
 int Liczba_s(lista_s Lista);
 int Wstaw(lista_s& Lista, dane Dana);
 int Usun(lista_s& Lista, dane Dana);
 void Pokaz_liste(lista_s& Lista, void(*p)(dane Dana));

//2. funkcje we/wy dla danych umieszczonych na liście symboli

 void Pokaz_dane (dane Dana);
 dane Dane(char* s);
```

```
//3. funkcje ogólnego przeznaczenia

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

 //4. elementy programu
const int Esc=27;
const int POZ=4;
char * Tab_menu[POZ] = { "1 : Wstawianie do listy symboli ",
                         "2 : Usuwanie z listy symboli",
                         "3 : Wydruk listy symboli",
                         " >Esc Koniec programu"};

//5.funkcje klienta korzystajace ze listy symboli
 void Wstaw_do_listy_s(lista_s& Slownik);
 void Usun_z_listy_s(lista_s& Slownik);

void main(void)
 {
    lista_s Slownik;
    char Wybor;

   clrscr();
   Inicjalizacja(Slownik,zero,Maks);
   do
   { Wybor= Menu(POZ, Tab_menu);
    switch (Wybor)
      {
           case '1' : Wstaw_do_listy_s(Slownik);
                      break;
        case '2' : Usun_z_listy_s(Slownik);
                      break;
        case '3' : Pokaz_liste(Slownik, Pokaz_dane);
                      break;
    }
   } while (Wybor !=Esc );
  }
```

```
//*********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 klienta korzystające z listy symboli***********
 void Wstaw_do_listy_s(lista_s& Slownik)
 {  dane  Dana= Dane("Podaj dane do wstawienia: ");
    if (Wstaw(Slownik, Dana))
     Komunikat("\nWstawiono do listy symboli");
    else  Komunikat("\nZly zakres symboli");
 }

void Usun_z_listy_s(lista_s& Slownik)
   { dane  Dana = Dane("Podaj dane do usuniecia z listy symboli: ");
     int d = Usun(Slownik,Dana);
     if  (d==1)   Komunikat("\n Usunieto z listy symboli");
     else
       if (d==2)  Komunikat("\n brak na liscie elementu z takim symbolem");
       else       Komunikat("\nSymbol poza zakresem");
    }

//******funkcje we/wy dla danych umieszczonych na liscie symboli*********
   dane Dane(char* s)
    { dane a;
     do
      { fflush(stdin);
        printf("\n\n%s",s);
      } while (scanf("%d",&a.klucz)!=1);
     return a;}

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

```
//*********interfejs ADT listy symboli************
inline int f_zero(dane d, const int c_zero)
 {  return d.klucz==c_zero;
 }

int Zmien(lista_s& Slownik, int delta)
  { dane* pom;
    pom = (dane*)realloc(Slownik.tab,sizeof(dane)*(Slownik.rozmiar+delta));
    if (pom)
        { Slownik.tab=pom;
        Slownik.rozmiar+=delta;
        }
    return pom!=NULL;
    }

 void Inicjalizacja(lista_s& Slownik, dane zero, int Maks)
   {Slownik.rozmiar=0;
    Slownik.zero=zero;
    Slownik.tab=NULL;
    if (!Zmien(Slownik,Maks)) exit(0);
    for (int i=0; i<Maks;i++)     //wstawianie symboli z pustymi kluczami
          Slownik.tab[i]=zero;
   }

 inline int Zakres(lista_s Slownik, int Klucz)
  { return (Klucz >=0 && Klucz<Slownik.rozmiar);}

 dane Szukaj_element(lista_s& Slownik, int Klucz)
  { return Slownik.tab[Klucz];}

int Wstaw(lista_s& Slownik, dane Dana)
   { if  (Zakres(Slownik, Dana.klucz))
       { Slownik.tab[Dana.klucz]=Dana;
         return 1;}
     else return 0;
   }
```

```
int Usun(lista_s& Slownik, dane Dana)
  {
    if  (Zakres(Slownik, Dana.klucz))
      if (!f_zero(Slownik.tab[Dana.klucz], c_zero))
        { Slownik.tab[Dana.klucz] = Slownik.zero;
          return 1;}
      else return 2;
    else return 0;
  }

 dane Wybierz(lista_s Slownik, int numer)
   {
     for (int i=0;i<Slownik.rozmiar;i++)
      if (Slownik.tab[i].klucz != Slownik.zero.klucz)
         if (numer-- == 0)
            return Slownik.tab[i];
     return Slownik.zero;
   }

 int Liczba(lista_s Slownik)
    { int ile=0;
      for (int i=0; i<Slownik.rozmiar; i++)
          if (Slownik.tab[i].klucz != Slownik.zero.klucz)
              ile++;
      return ile;
    }

    void Pokaz_liste(lista_s& Slownik, void(*p)(dane Dana))
     {
       for (int i=0;i<Slownik.rozmiar;i++)
           if (!f_zero(Slownik.tab[i], c_zero))
              p(Slownik.tab[i]);  //wyświetlanie elementów z niepustym kluczem
     }
```