1. Abstrakcyjne typy danych

1.1. Główne cechy typów danych stosowanych w programach:

1) musza byc dostosowane do rozwiazywanego problemu
2) musza zawierac dwa rodzaje informacji:
   2.1) zbiór wlasnosci
   2.2) zbiór dzialan.

Np. typ **int**
wlasnosci: reprezentuje liczby calkowite o wartosciach od -32768 do +32767 zakodowanych w kodzie dwójkowym na dwóch bajtach
dzialania: zmiana znaku, dodawanie, mnozenie, dzielenie, modulo...

1.2. Trzy etapy procesu definiowania typów przez programiste [wg Stephen Prata - Szkola programowania, Jezyk C]:

1) Przygotowanie opisu ADT (abstrakcyjnego typu danych).

   Abstrakcyjny opis obejmuje wlasnosci typu i operacji, jakie mozna na nim wykonac. Opis ten nie powinien byc zwiazany z zadna konkretna implementacja i jezykiem programowania. Taka formalna charakterystyka nosi nazwe abstrakcyjnego typu danych ADT.

2) Opracowanie interfejsu programistycznego realizujacego ADT -

   Jest to wskazanie sposobu przechowywania danych i opisanie zbioru funkcji wykonujacych potrzebne operacje. W jezyku C/C++ etap ten moze polegac na utworzeniu definicji struktury i prototypów funkcji przetwarzajacych. Funkcje te pelnia dla nowego typu te sama role, co operatory w przypadku podstawowych typów jezyka C/C++. Utworzony w ten sposób interfejs bedzie stosowany przez osoby, które chca skorzystac z nowego typu danych

3) Pisanie kodu implementujacego interfejs

   Ten krok jest kluczowy, w którym nalezy szczególowo zrealizowac wymagania wynikajace z opisu. Nalezy zauwazyc, ze programista korzystajacy z interfejsu nie musi juz orientowac sie w szczególach implementacji

Wniosek: Utworzony typ danych, definiowany przez programiste stanowi pewien kompletny element jezyka, który moze byc uzywany wielokrotnie w programach. Jedyny wlasciwy sposób wykorzystania nowego typu odbywa sie za posrednictwem funkcji z interfejsu typu. Nie nalezy bezposrednio odwolywac sie do zmiennych reprezentujacych dana zdefiniowanego typu.

2.1. Definiowanie listy nieuporzadkowanej jako statycznej tablicy struktur metoda ADT

Etap 1 - Opis ADT

Nazwa typu - Statyczna lista nieuporzadkowanych elementów
Wlasnosci typu: Potrafi przechowac ciag elementów o ograniczonym rozmiarze
Dostepne dzialania:

- Inicjalizacja listy
- Okreslenie, czy lista jest pelna
- Okreslenie, czy lista jest pusta
- Wyszukanie elementu
- Dodanie elementu wewnatrz, na poczatku i na koncu listy,
- Usuwanie elementu wewnatrz, na poczatku i na koncu listy,
- Przejscie przez liste i przetwarzanie kazdego elementu
- Przetwarzanie wyszukanego elementu

Etap 2 - Budowa interfejsu

```
void Inicjalizacja (int & ile);
/* dzialanie: inicjuje liste
   warunki wstepne: lista nie jest zainicjowana
   warunki koncowe: lista zostaje zainicjowana jako pusta, ile jest równe 0 i jest
        numerem elementu pustego. Dopiero po wywolaniu tej funkcji prawdziwe sa
        definicje pozostalych funkcji */

int Pusta (int ile);
/*dzialanie: okresla czy lista jest pusta
   warunki wstepne: ile jest numerem ostatniego elementu wstawionego do listy
   warunki koncowe: funkcja zwraca wartosc 1, gdy lista jest pusta, w przeciwnym
        wypadku zwraca 0*/

int Pelna (int ile);
/*dzialanie: okresla czy lista jest pelna
   warunki wstepne: ile jest numerem ostatniego elementu wstawionego do listy
   warunki koncowe: funkcja zwraca wartosc 1, gdy lista jest pelna, w przeciwnym
        wypadku zwraca 0*/

int Szukaj(int ile, int ktory);
/*dzialanie: szuka wskazanego elementu
   warunki poczatkowe: który jest numerem szukanego elementu, ile jest numerem
        ostatniego elementu przy usuwaniu lub wiekszym o 1 przy wstawianiu
   warunki koncowe: funkcja zwraca 1, gdy znaleziono element o numerze ktory
        spelniajacy warunki 1<=ktory<=ile, w przeciwnym wypadku zwraca 0 */
```

void Wstaw(OSOBA tab[], OSOBA dane, **int** ktory, **int** &ile);
/**dzialanie:* dodaje element na poczatku, wewnatrz lub na koncu listy
warunki poczatkowe: ile jest numerem ostatnio wstawionego elementu do listy lub elementu pustego, *w*ynik funkcji *Pelna* po podstawieniu wartosci *ile* jest równy 0, *który* jest numerem miejsca do wstawienia podanym przez funkcje *Szukaj* z wynikiem 1 dla *ile=ile+1*, który oznacza numer miejsca za ostatnio wstawionym elementem lub pustym
warunki koncowe: funkcja dodaje element na wskazanym miejscu *ktory* po rozsunieciu elementów lub na koncu ciagu, zwieksza o 1 numer ostatniego elementu *ile* */

void Usun(OSOBA tab[], **int** ktory, **int** &ile);
/**dzialanie:* usuwa element nalezacy do listy
warunki poczatkowe: wynik funkcji *Pusta* po podstawieniu wartosci *ile* jest równy 0, *ile* jest numerem ostatniego elementu wstawionego do listy, *który* jest numerem elementu do usuniecia podanym przez funkcje *Szukaj* z wynikiem 1,
warunki koncowe: funkcja usuwa element na wskazanym miejscu *ktory* przez zsuniecie lub ostatni element ciagu, zmniejsza liczbe elementów w *ile* */

void Usun_tablice(**int** &ile);
/**dzialanie:* kasuje liczbe elementów i inicjuje tablice
warunki poczatkowe: tablica jest pusta lub niepusta, *ile* oznacza numer ostatniego elementu wstawionego do listy lub elementu pustego
warunki koncowe: ile jest numerem pustym, równym 0*/

void Dla_kazdego(OSOBA tab[], **int** ile, zrob);
/**dzialanie:* wykonuje funkcje na kazdym wstawionym elemencie do listy
warunki poczatkowe: wynik funkcji *Pusta* po podstawieniu wartosci *ile* jest równy 0, *ile* jest numerem ostatniego elementu wstawionego do listy, *zrob* jest typem funkcji, która przetwarza element listy i nie zwraca wartosci.
warunki koncowe: funkcja typu *zrob* jest wykonywana tylko raz dla kazdego elementu wstawionego do listy*/

void Dla_jednego(OSOBA tab[], **int** ile, **int** ktory, zrob);
/**dzialanie*: wykonuje funkcje na wybranym elemencie wstawionym do listy
*warunki poczatkowe: w*ynik funkcji *Pusta* po podstawieniu wartosci *ile* jest równy 0, *ile* jest numerem ostatniego elementu wstawionego do listy, *zrob* jest typem funkcji, która przetwarza element listy o numerze *ktory* podanym przez funkcje *Szukaj* z wynikiem 1 i nie zwraca wartosci.
warunki koncowe: funkcja typu *zrob* jest wykonywana tylko raz dla elementu o numerze *ktory**/

Etap 3 - Implementacja

Definicja typu OSOBA w pliku naglówkowym *strukt.h*

```
#ifndef STRUKT
   #define STRUKT
   const int DL=10;
   struct OSOBA
       {
        int Numer;
        char Nazwisko[DL];
       };
#endif
```

Zawartosc pliku naglówkowego *we_wy.h* zawierajacego deklaracje uniwersalnych funkcji we/wy dla struktur typu OSOBA

```
#ifndef WE_WY
   #define WE_WY
   #include "strukt.h"
   void Pokaz_dane (OSOBA &Dana);
   OSOBA Dane();
#endif
```

Zawartosc pliku modulowego *we_wy.cpp* zawierajacego definicje uniwersalnych funkcji we/wy dla struktur typu OSOBA

```
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include "we_wy.h"
#include "strukt.h"
 OSOBA Dane()
  {char bufor[DL+2];
   OSOBA Nowy;
   bufor[0]=DL;
   cprintf("\r\nnumer: ");
   cgets(bufor);
   Nowy.Numer=atoi(bufor+2);
   cprintf("\r\nnazwisko: ");
   strcpy(Nowy.Nazwisko,cgets(bufor));
   return Nowy;}
 void Pokaz_dane(OSOBA &Dana)
  {cprintf("\r\nNumer: %d\r\n", Dana.Numer);
   cprintf("Nazwisko:       %s\r\n", Dana.Nazwisko);
   cprintf("Nacisnij dowolny klawisz...\r\n"); getch();}
```

Zawartosc pliku naglówkowego *tablica1.h* z deklaracjami funkcji interfejsowych defniowanej statycznej tablicy struktur jako listy nieuporzadkowanej

```
#ifndef TABLICA
    #define TABLICA
    #include "strukt.h"
    const int ROZ=5;
    typedef void(*zrob)(OSOBA &);
    void  Inicjalizacja(int& ile);
    int   Pelna(int ile);
    int   Pusta(int ile);
    int   Szukaj(int ile, int ktory);
    void  Wstaw(OSOBA*tab, OSOBA dane, int ktory, int &ile);
    void  Usun(OSOBA*tab, int ktory, int &ile);
    void  Usun_tablice(int &ile);
    void  Dla_kazdego(OSOBA*tab, int ile, zrob);
    void  Dla_jednego(OSOBA*tab, int ktory, zrob);
#endif
```

Zawartosc pliku modulowego *tablica1.cpp* z definicjami funkcji interfejsowych

```
#include "tablica1.h"
void Inicjalizacja(int & ile)   {ile=0;}
int Pelna(int ile)  {return ile==ROZ;}     //wy=1 tablica pelna, wy=0 jest miejsce
int Pusta(int ile)  {return ile==0;}       //wy=1 tablica pusta, wy=0 sa elementy
int Szukaj(int ile, int ktory)
  {return ktory>=1 && ktory<=ile;}         // wy-1 znalazl; wy = 0 nie
void Wstaw(OSOBA* tab, OSOBA dane, int ktory,int &ile)
  { for (int i=ile; i>=ktory; i--)         // zalozenia: 0<=ile <N  oraz 1<=ktory<=ile+1
      tab[i]=tab[i-1];
    tab[ktory-1]=dane;
    ile++;}
void Usun(OSOBA* tab, int ktory, int &ile)
   { for (int i=ktory-1; i<ile-1; i++)     // zalozenia: 0<ile <=N i 1<=ktory<=ile
      tab[i]=tab[i+1];
    ile--;}
void Usun_tablice (int &ile)    { Inicjalizacja(ile); }
void Dla_kazdego(OSOBA* tab, int ile, zrob funkcja)
   {for (int i=0; i<ile;i++)  funkcja(tab[i]); }  //wykonaj operacje zrob na zawartosci tablicy
void Dla_jednego(OSOBA* tab, int ktory, zrob funkcja)
   { funkcja(tab[ktory-1]);}                  //wykonaj czynnosc zrob dla elementu tablicy
```

Przykladowa aplikacja wykorzystujaca liste nieuporzadkowana reprezentowana przez statyczna tablice struktur (*tablist1.cpp*)

```
#include <conio.h>
#include <stdlib.h>
#include "tablica1.h"          //definicja statycznej tablicy struktur jako listy nieuporzadkowanej
#include "dodatki.h"           //uniwersalne funkcje obslugujace menu i komunikaty
#include "we_wy.h"             //uniwersalne funkcje we/wy dla struktur typu OSOBA

 char *Polecenia[POZ]={"Tablica OSOBA tab[Roz] - obsluga typu lista",
                       " Nacisnij:",
                       " 1 - aby wstawic element do listy",
                       " 2 - aby usunac element z listy",
                       " 3 - aby wyswietlic wybrany element z listy",
                       " 4 - aby wyswietlic liste, ",
                       " 5 - aby usunac liste",
                       " 6 - aby zakonczyc prace."};

// funkcje posrednio przetwarzajace tab i ile za pomoca funkcji z modulu tablica1
 void Wstaw_(OSOBA*tab,int &ile);
 void Usun_(OSOBA*tab, int &ile);
 void Pokaz_jeden(OSOBA*tab,int ile);
 void Pokaz_(OSOBA*tab,int ile);
// funkcja ogólnego przeznaczenia
 int Losuj(int zakres);

 void main(void)
  {int ile;
   OSOBA tab[ROZ];
   char Co;
   randomize();
   Inicjalizacja(ile);
    do
     {Co = Menu(POZ,Polecenia);
      switch(Co)
       {case '1' : Wstaw_(tab,ile); break;
            case '2' : Usun_(tab,ile); break;
            case '3' : Pokaz_jeden(tab,ile); break;
            case '4' : Pokaz_(tab,ile); break;
            case '5' : Usun_tablice(ile); break;
            default  : Komunikat("\r\nKoniec programu");
       }
     }while (Co < '6' && Co>'0'); }
```

```
int Losuj(int zakres)                          //model mylacego sie uzytkownika programu
   { int ktory=random(zakres);
     cprintf("\r\n\nNumer elementu: %d",ktory); getch();
     return ktory; }


void Pokaz_jeden(OSOBA* tab,int ile)           //obsluga wyszukania
   {
    if (Pusta(ile)) Komunikat("\r\nTablica pusta");
    else
     {int ktory=Losuj(ile+4);
      if (Szukaj(ile,ktory)) Dla_jednego(tab,ktory,Pokaz_dane);
      else Komunikat("\r\nPodano zly numer"); } }


  void Wstaw_(OSOBA*tab,int &ile)              //obsluga wstawiania
   {
     if (Pelna(ile)) Komunikat("\r\nPelna tablica");
     else
     {int ktory=Losuj(ile+4);
      if (!Szukaj(ile+1,ktory)) Komunikat("\r\nPodano zly numer");
      else
            {cprintf("\r\nPodaj tab[%d]",ktory-1);
             Wstaw(tab,Dane(),ktory,ile); } } }


  void Usun_(OSOBA* tab,int &ile)              //obsluga usuwania
   {
     if (Pusta(ile)) Komunikat("\r\nTablica pusta");
     else
      { int ktory=Losuj(ile+4);
           if (!Szukaj(ile,ktory)) Komunikat("\r\nPodano zly numer");
           else
            { Usun(tab,ktory,ile);
              Komunikat("\r\nUsunieto element"); } } }


  void Pokaz_(OSOBA* tab,int ile)              //obsluga wyswietlenia zawartosci tablicy
   {
    if (Pusta(ile)) Komunikat("\r\nTablica pusta");
    else Dla_kazdego(tab,ile,Pokaz_dane);
   }
```

Zawartosc pliku naglówkowego *dodatki.h* z pomocniczymi funkcjami obslugujacymi menu programu oraz komunikaty

```
#ifndef DODATKI
    #define DODATKI
    const int POZ=8;
    void Komunikat(char*);
    char Menu(const int ile, char *Polecenia[POZ]);
#endif
```

Zawartosc pliku modulowego *dodatki.cpp* z pomocniczymi funkcjami obslugujacymi menu programu oraz komunikaty

```
#include <conio.h>
#include "dodatki.h"

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

void Komunikat(char* s)
{
  cprintf(”%s”,s);  getch();
}
```