#include<stdio.h>

#include<malloc.h>

typedef struct zespolona

{

float re, im;

} zespolona;

typedef struct element

{

zespolona liczba;

struct element\* nastepny;

struct element\* poprzedni;

} element;

element\* nowy\_elem(element\* glowa, zespolona liczba)

{

element\* n = malloc(sizeof(element));

\*n = (element) { .liczba = liczba, .nastepny = NULL };

if (glowa)

{

for(element\* ptr = glowa; ptr->nastepny || !(ptr->nastepny = n); ptr = ptr->nastepny);

return glowa;

}

return n;

}

element\* ile(element\*glowa)

{

if (!glowa) //pusta lista

return 0;

int i=1;

while (glowa->nastepny)

{

glowa=glowa->nastepny;

i++;

}

return i;

}

element\* usun\_ostatni(element\* glowa)

{

if(!glowa) return; //pusta lista

if(!glowa ->nastepny) //jeden element

{

free(glowa);

return;

}

//wiele elementow

while(glowa->nastepny && glowa-> nastepny->nastepny)

glowa = glowa ->nastepny;

free(glowa->nastepny);

glowa ->nastepny = NULL;

}

element\* usun\_ity(element\* glowa, int index)

{

element\* poprzedni = NULL;

for(int i = 1 ; glowa && glowa -> nastepny && i++<index ; )

{

poprzedni = glowa;

glowa = glowa -> nastepny;

}

poprzedni -> nastepny = glowa ->nastepny;

free(glowa);

}

element \*nowy\_po(element \*glowa, int index, zespolona liczba)

{

element \*glowa\_zero = glowa; //zapisuje adres glowy zerowej i potem ja funkcja zwraca

if(glowa)

{

int dlugosc = ile(glowa);

if (index >= dlugosc)

glowa = nowy\_elem(glowa, (zespolona ) {}); // dodaje na koncu

else //jezeli lista zawiera elementy i podany index nie jest wiekszy od dlugosci listy

{

element\* nowy = malloc(sizeof(element)); //nowy element

\*nowy = (element) { .liczba = liczba, .nastepny = NULL , .poprzedni = NULL };

if (index == 0) // Wstawienie poczatek listy

{

nowy->nastepny = glowa; //laczenie komorek listy

glowa->poprzedni = nowy;

glowa\_zero = nowy; //i wpisanie w komorke zerowa

}

else

{

for (int i = 1; i < index && glowa; i++, glowa = glowa->nastepny); //szukamy index

nowy->nastepny = glowa->nastepny; //laczy komórke na miejscu index+1 z komorka nowy

if (glowa->nastepny)

{

glowa->nastepny->poprzedni = nowy; //i w druga strone

}

nowy->poprzedni = glowa; //laczy komórke na miejscu index z komórka nowy

glowa->nastepny = nowy; //i w druga strone

}

}

}

return glowa\_zero;

}

void wypisz(element\* glowa)

{

if(!glowa) //pusta lista

return;

printf(" liczba zespolona = %f i %f\n", glowa->liczba.re , glowa->liczba.im);

if(glowa->nastepny)

wypisz(glowa->nastepny);

}

element \*zniszcz(element\* glowa) // czysci i zwalnia pamiec calej listy

{

if(glowa) //czy nie jest pusta

{

zniszcz(glowa->nastepny);

free(glowa);

}

return NULL;

}

void main()

{

element\* glowa =NULL;

glowa = nowy\_elem(glowa, (zespolona) {1, 1});

glowa = nowy\_elem(glowa, (zespolona) {-3.1415, 2.71});

glowa = nowy\_elem(glowa, (zespolona) {-4, -2.71});

glowa = nowy\_elem(glowa, (zespolona) {-5, -2.71});

glowa = nowy\_elem(glowa, (zespolona) {-8, -2.71});

glowa = nowy\_elem(glowa, (zespolona) {-9, -2.71});

wypisz(glowa);

printf("dlugosc = %d\n",ile(glowa));

usun\_ostatni(glowa);

usun\_ity(glowa,3);

nowy\_po(glowa, 3 ,(zespolona) {3 ,4});

wypisz(glowa);

zniszcz(glowa);

}

//dodawanie wyswietlanie usuwanie dlugosc