NAPISY

Napisz funkcję godzina, która dostaje w argumentach trzy liczby całkowite godz, min i sek i zwraca jako wartość napis w formacie godz:min:sek.

```
char* godzina(int godz, int min, int sek)
{
   char* czas = malloc(sizeof(char) * 9);
   sprintf(czas, "%02d:%02d:%02d", godz, min,
sek);
   return czas;
}

int main() {
   int godz = 9, min = 45, sek = 12;
   printf("%s\n", godzina(godz, min, sek));
   return 0;
}
```

Konwersja char int

```
int konwertujNaLiczbe(char* napis) {
   int liczba = 0;
   int i = 0;
   while (napis[i] != '\0') {
      if (napis[i] >= '0' && napis[i] <= '9') {
         liczba = liczba * 10 + (napis[i] - '0');
////napis[i] - '0' zamienia znak 'cyfra' na cyfre w
dziesietnym///
         i++;
      }
      else {
         return 0;  // Jeśli napotkano znak inny
niż cyfra, zwróć zero
      }
   }
   return liczba;
}

int main()
{
   char* napis="35a";
   printf("%d\n",konwertujNaLiczbe(napis));
   return 0;
}
```

Napisz funkcję wytnij, która wycina z otrzymanego napisu znaki o indeksach od n do m (n <= m.). Otrzymany w argumencie napis może mieć dowolną liczbę znaków (w tym mniejszą od n lub m.)

```
#void wytnij(char*napis, int n, int m)
{
   int d;
   while(napis[d]){
      d++;
   }
   if(m>=d){
      m=d-1;
   }
   for(int i=0; i<m-n+1; i++){
      napis[i]=napis[n+i];
   }
   napis[m-n+1] = '\0';
}

int main()
{
   char nap[] = "Olsztyn";
   wytnij(nap,3,5);
   printf("%s\n", nap);
   return 0;
}
```

Funkcje znakowe

1. `isalnum` - sprawdza, czy znak jest alfanumeryczny (cyfra lub litera).
2. `isalpha` - sprawdza, czy znak jest literą (alfabetycznym).
3. `islower` - sprawdza, czy znak jest małą literą.
4. `isupper` - sprawdza, czy znak jest dużą literą.
5. `isdigit` - sprawdza, czy znak jest cyfrą.

tablica char i wskaznik na char

```
int main()
{
   char modyf[] = „tekst”;
   modyf[4] = ’M’;
   char *niemodyf = ”tekst”;
   return 0;
}
```

Funkcje napisy

```
void wyczysc(char *napis){
   napis[0] = '\0';
}

int dlugosc(char*napis)
{
   int temp=0;
   while(*(napis++))
   {
      temp++;
   }
   return temp;
}

int dlugosc2(char napis[])
{
   int temp=0;
   for(int i=0;napis[i]!='\0';i++)
   {
      temp++;
   }
   return temp;
}
```

strlen i sizeof

```
int main()
{
   char nap1[] = "Hello World";
   char nap2[50] = "Hello World";
   printf("%lu\n",sizeof nap1); // rozmiar tablicy nap1, który wynosi 12 bajtów (11 znaków + znak null)
   printf("%lu\n",strlen(nap1)); //zwraca długość łańcucha nap1, która wynosi 11 znaków
   printf("%lu\n",sizeof nap2); //rozmiar tablicy nap2, który wynosi 50 bajtów
   printf("%lu\n",strlen(nap2)); //długość łańcucha nap2, która wynosi 11 znaków (ignoruje nadmiarową przestrzeń w tablicy, ponieważ nie jest używana)
   return 0;
}
```

Napisz funkcję wytnijzw, która dostaje jako argument dwa napisy nap1 i nap2, i wycina z napisu nap1 wszystkie znaki występującetakże w napisie nap2.

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

void wytnijzw(char* nap1, char* nap2)
{
   int d1 = 0;
   while(nap1[d1])
   {
      d1++;
   }
   int d2 = 0;
   while(nap2[d2])
   {
      d2++;
   }
   for (int i=0; i<d2; i++)
   {
      for(int j=0; j<d1; j++)
      {
         if(nap1[j] == nap2[i])
         {
            for(int k=j; k<d1-1; k++)
            {
               nap1[k] = nap1[k+1];
            }
            nap1[d1-1] = '\0';
            d1--;
            j--;
         }
      }
   }
}

int main()
{
   char nap1[] = "gruszka";
   char nap2[] = "truskawka";
   wytnijzw(nap1, nap2);
   printf("%s\n", nap1);
   return 0;
}
```

TABLICE TABLIC

- **każdy wymiar tablicy tworzony przez malloca**
- **dynamiczne (?)**
- **o zmiennym rozmiarze**
- *** wskaźniki**

Suma na nieparzystych indeksach

```
int foo(int **tab,int n, int m)
{
   int suma=0;
   for(int i=0; i<n; i++)
      for(int j=0; j<m.; j++){
         if(i%2==1 && j%2==1){
            suma+=tab[i][j];
         }
      }
      return suma;
}
int main()
{
   int **tab=malloc(4*sizeof(int*));
   tab[0]=malloc(3*sizeof(int));
   tab[1]=malloc(3*sizeof(int));
   tab[2]=malloc(3*sizeof(int));
   tab[3]=malloc(3*sizeof(int));
   tab[0][0]=1;
   tab[0][1]=2;
   tab[0][2]=3;
   tab[1][0]=4;
   tab[1][1]=5;
   tab[1][2]=6;
   tab[2][0]=7;
   tab[2][1]=8;
   tab[2][2]=7;
   tab[3][0]=6;
   tab[3][1]=4;
   tab[3][2]=2;
   printf("%d\n",foo(tab,4,3));
   return 0;
}
```

Napisz funkcję, która tworzy dynamiczną dwuwymiarową tablicę tablic elementów typu int o wymiarach n na m, i zwraca jako wartość wskaźnik do niej.

```
int ** alokuj(int n, int m)
{
   int ** temp = malloc(n*sizeof(int*));
   for(int i=0; i<n; i++){
      temp[i]=malloc(m*sizeof(int*));
   }
   return temp;
}

void zwolnij(int **tab, int n, int m)
{
   for(int i=0; i<n; i++){
      free(tab[i]);
   }
   free(tab);
}

int main()
{
   int ** tab = alokuj(2,3);
   printf("%p\n", tab);
   zwolnij(tab, 2,3);
   return 0;
}
```

Napisz funkcję, tworzy dynamiczną dwuwymiarową trójkątną tablicę tablic o wymiarach n na n i zwraca jako wartość wskaźnik do niej.

```
int* foo(int n)
{
   int ** temp = malloc(n*sizeof(int*));
   int j=1;
   for (int i =0; i<n; i++)
   {
      temp[i] = malloc(j*sizeof(int));
      j++;
   }
   return temp;
}
```

Napisz funkcję, która dostaje jako argumenty dwuwymiarową tablicę tablic o elementach typu int oraz jej wymiary, i zmienia kolejność kolumn w tablicy w taki sposób, że kolumna pierwsza ma się znaleźć na miejscu drugiej, kolumna druga ma się znaleźć na miejscu trzeciej itd., natomiast ostatnia kolumna ma się znaleźć na miejscu pierwszej (przyjmujemy, że dwa elementy tablicy leżą w tej samej kolumnie, jeżeli mają taką samą drugą współrzędną).

```
void wyswietl(int **tab, int n, int m)
{
   for(int i=0; i<n; i++){
      for(int j=0; j<m; j++){
         printf("[%d][%d]=[%d]", i,j,tab[i][j]);
      }
      printf("\n");
   }
   printf("----------\n");
}
```

```
void foo(int**tab, int n, int m)
{
   int temp;
   for(int i=0; i<n; i++)
   {
      temp = tab[i][m-1];
      for(int j=m-1; j>0; j--)
      {
         tab[i][j] = tab[i][j-1];
      }
      tab[i][0] = temp;
   }
}
```

TABLICE WIELOWYMIAROWE (ELEMENTÓW)

- **Do każdego wymiaru są nawiasy kwadratowe**
- **o stałym rozmiarze**
- **statyczne (?)**
- **inaczej macierze**
- **typ nazwa[wymiar1][wymiar2]...**

Wyświetl tablicę elementów

```
void wyswietl(int n, int m, int tab[n][m])
{
   for(int i=0; i<n; i++)
   {
      for(int j=0; j<m; j++)
      {
         printf("[%d,%d]=%d", i,j,tab[i][j]);
      }
      printf("\n");
   }
}
```

Napisz funkcję, która dostaje w argumentach tablicę dwuwymiarową elementów typu int, o pierwszym wymiarze podanym jako drugi argument funkcji oraz drugim wymiarze równym 100, która to funkcja zwraca jako wartość sumę wartości elementów tablicy.

```
int suma(int tab[][100], int n)
{
   int suma = 0;
   for (int i = 0; i < n; i++){
      for (int j = 0; j < 100; j++){
         suma += tab[i][j];
      }
   }
   return suma;
}

int main()
{
   int tablica[20][100];
   for (int i = 0; i < 20; i++){
      for (int j = 0; j < 100; j++){
         tablica[i][j] = i;
      }
   }
   printf("Suma= %d\n",suma(tablica,20));
   return 0;
}
```

STRUKTURY

A. Napisz strukturę `Osoba` z polami `imie` (tablica znaków długości 20) oraz `wiek` (typu `int`). Następnie napisz funkcje i wywołają każdą z nich co najmniej jeden raz:

a) `initOsoba` - funkcja przyjmuje dwa argumentem imię i wiek i zwraca nowo-utworzoną strukturę ustawiającą składowe z przekazanych argumentów.
b) `pokazOsoba` - funkcja, której argumentem jest zmienna w typie `Osoba`. Funkcja ma wypisać opis przekazanego argumentu (wpisać wiek i imię na standardowym wyjściu).
c) `urodziny` - funkcja, której argumentem jest wskaźnik do struktury typu `Osoba`. Funkcja ma powiększyć wiek o 1 w przekazanym argumencie.

```
struct Osoba
{
   char imie[20];
   int wiek;
};

struct Osoba initOsoba(char im[50], int wk)
{
   struct Osoba temp;
   temp.wiek = wk;
   //strcpy(temp,imie,im);
   int i=0;
   for(i=0; im[i] != 0; i++)
   {
      temp.imie[i] = im[i];
   }
   temp.imie [i] = 0;
   return temp;
};

void pokazOsoba(struct Osoba arg)
{
   printf("Imie: %s, wiek: %d\n", arg.imie,arg.wiek);
}

void urodziny(struct Osoba *wsk)
{
   wsk->wiek++;
}

int main()
{
   struct Osoba nowa= initOsoba("Patryk", 25);
   pokazOsoba(nowa);
   urodziny(&nowa);
   return 0;
}
```

Trójkąt

```
struct trojkat
{
   float a,b,c;
};

float obwod(struct trojkat arg)
{
   return arg.a + arg.b + arg.c;
}

void przepisz(struct trojkat troj1, struct trojkat *troj2)
{
   *troj2=troj1;
}

int main()
{
   struct trojkat tr = {3,4,5};
   printf("%f\n", obwod(tr));
   return 0;
}
```

Student

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

struct student{
   char imie[50];
   int numer_albumu;
};
int dlugosc(char *napis)
{
   int i=0;
   while(napis[i]!='\0'){
      i++;
   }
   return i;
}
int foo(struct student tab[],int n){
   int indeks=0;
   int krotszy=dlugosc(tab[0].imie);
   for(int i=0;i<n;i++){
      if(dlugosc(tab[i].imie)<krotszy)
      {
            krotszy=dlugosc(tab[i].imie);
            indeks=i;
      }
   }
   return tab[indeks].numer_albumu;

}
int main()
{
   struct student tab[6];
   strcpy(tab[0].imie,"aaa");
   tab[0].numer_albumu = 11;
   strcpy(tab[1].imie,"aaaa");
   tab[1].numer_albumu = 22;
   strcpy(tab[2].imie,"aaaaa");
   tab[2].numer_albumu= 33;
   strcpy(tab[3].imie,"aaaa");
   tab[3].numer_albumu = 4;
   strcpy(tab[4].imie,"a");
   tab[4].numer_albumu = 55;
   strcpy(tab[5].imie,"aaaa");
   tab[5].numer_albumu = 66;

   printf("%d",foo(tab,6));
   return 0;
}
```

Liczba zespolona

```
struct Zespolone
{
   double urojona;
   double rzeczywista;
};

struct Zespolone initZespolone(double im, double re)
{
   struct Zespolone temp;
   temp.urojona = im;
   temp.rzeczywista = re;
   return temp;
};

struct Zespolone dodaj(struct Zespolone Z1, struct Zespolone Z2)
{
   struct Zespolone suma;
   suma.urojona = Z1.urojona + Z2.urojona;
   suma.rzeczywista = Z1.rzeczywista + Z2.rzeczywista;
   return suma;
}

int main()
{
   struct Zespolone z1 = initZespolone(5.2, 7.8);
   struct Zespolone z2 = initZespolone(3.4,2.11);
   struct Zespolone wynik = dodaj(z1, z2);
   printf("Suma = %.2lf + %.2lfi\n", wynik.rzeczywista, wynik.urojona);
   return 0;
}
```

```
//ALBO
//tab[0] = (struct student) { "aa", 11 };
//tab[1] = (struct student) { "aaaa", 22 };
//tab[2] = (struct student) { "aaaaa", 33 };
//tab[3] = (struct student) { "aaaa", 44 };
//tab[4] = (struct student) { "a", 55 };
//tab[5] = (struct student) { "asdasa", 66 };
```

Zdefiniuj strukturę punktn służącą do przechowywania współ rzędnych punktów w n-wymiarowej przestrzeni kartezjańskiej. Do przechowywania poszczególnych wymiarów wykorzystaj tablicę n-elementową. W strukturze punktn przechowuj także ilość wymia rów przestrzeni.
Napisz funkcję, która otrzymuje jako argumenty tablice tab1 i tab2 o argumentach typu struct punktn oraz ich wspólny rozmiar, i prze pisuje zwartość tablicy tab1 do tablicy tab2. Zakładamy, że tablica tab2 jest pusta.

ENUM
UNION

```
struct punktn
{
   int n;
   int * wspolrzedne;
};

void przepisz(struct punktn tab1[], struct punktn tab2[], int n)
{
   for(int i=0;i<n;i++){
      tab2[i].n = tab1[i].n;
      tab2[i]. wspolrzedne = malloc(tab2[i].n * sizeof(int));
      for(int j=0;j<tab1[i].n;j++){
         tab2[i].wspolrzedne[j] = tab1[i].wspolrzedne[j];
      }
   }
}

void wyswietl(struct punktn tab[], int n)
{
   for(int i=0;i<n;i++)
   {
      printf("[%d]: ", i);
      for(int j=0;j<tab[i].n;j++){
         printf("%d ", tab[i].wspolrzedne[j]);
      }
      printf("\n");
   }
}

int main()
{
   struct punktn p1;
   p1.n = 4;
   p1.wspolrzedne = malloc(p1.n *sizeof(int));
   p1.wspolrzedne[0] = 8;
   p1.wspolrzedne[1] = -3;
   p1.wspolrzedne[2] = 2;
   p1.wspolrzedne[3] = 11;
   struct punktn p2;
   p2.n = 5;
   p2.wspolrzedne = (int []) {3,-2,3,4,-9};
   struct punktn tab1[] = {p1,p2};
   wyswietl(tab1,2);
   struct punktn tab2[] =
   {{5, (int []) {6,7,-23,2,2}},{1, (int []) {9}}};
   return 0;
}
```

```
union Liczba
{
   int calkowita;
   double wymierna;
};

struct Dane
{
   int tp;
   union Liczba zaw;
};

struct Dane wczytaj()
{
   struct Dane dane;
   printf("liczba calkowita: wpisz 0\n");
   printf("liczba wymierna: wpisz 1\n");
   scanf("%d", &dane.tp);
   if(dane.tp == 0){
      printf("podaj liczbe calkowita: ");
      scanf("%d", &dane.zaw.calkowita);
   }
   else{
      printf("podaj liczbe wymierna: ");
      scanf("%lf", &dane.zaw.wymierna);
   }
   return dane;
};

int main()
{
   struct Dane d = wczytaj();
   return 0;
}
```

```
enum zwierzak {pies, kot, chomik};

int main()
{
   enum zwierzak zmienna = chomik;
   printf("%d\n", zmienna);
   printf("%Iu\n", sizeof(enum zwierzak));
   return 0;
}
```

LISTA KIERUNKOWA

```
#include <stdio.h>
#include <stdlib.h>
struct node{
   int t;
   struct node * next;
};
int foo(struct node *lista)
{
   int last_ujemny=NULL;
   struct node *current=lista;
   while(current!=NULL){
      if(current->t<0)
         last_ujemny=current->t;
///current->t to wartosc// current to wskaźnik na wartosc////
      current=current->next;
   }
   return last_ujemny;

}
int main()
{
   struct node *lista=malloc(sizeof(struct node));
   lista->next=malloc(sizeof(struct node));
   lista->next->t=1;
   lista->next->next=malloc(sizeof(struct node));
   lista->next->next->t=-2;
   lista->next->next->next=malloc(sizeof(struct node));
   lista->next->next->next->t=-4;
   lista->next->next->next->next=malloc(sizeof(struct node));
   lista->next->next->next->next->t=3;
   lista->next->next->next->next->next=NULL;

   printf("%d\n",foo(lista));
   return 0;
}
```

INICJALIZACJA LISTY BEZ GŁOWY

```
struct element *lista = malloc(sizeof(struct element));
  lista->i = 1;
  lista->next = malloc(sizeof(struct element));
  lista->next->i = 2;
  lista->next->next = malloc(sizeof(struct element));
  lista->next->next->i = 3;
  lista->next->next->next = NULL;
```

INICJALIZACJA LISTY Z GŁOWĄ

```
struct element *lista = malloc(sizeof(struct element));
   lista->next = malloc(sizeof(struct element));
   lista->next->i = 1;
   lista->next->next = malloc(sizeof(struct element));
   lista->next->next->i = -3;
   lista->next->next->next = malloc(sizeof(struct element));
   lista->next->next->next->i = 5;
   lista->next->next->next->next = NULL;
```

LISTY BEZ GŁOWY

```
struct element
{
   int i;
   struct element *next;
};

void zeruj(struct element *lista)
{
   struct element * wsk = lista;
   while(wsk != NULL){
      wsk->i = 0;
      wsk = wsk->next;
   }
}

void wyswietlListeBezGlowy(struct element *lista)
{
   if (lista == NULL)
   {
      printf("Lista jest pusta.\n---\n");
      return;
   }
   struct element *wsk = lista;
   while (wsk != NULL){
      printf("%d\n", wsk->i);
      wsk = wsk->next;
   }
   printf("---\n");
}

int main()
{
   struct element *lista = malloc(sizeof(struct element));
   lista->i = 1;
   lista->next = malloc(sizeof(struct element));
   lista->next->i = 2;
   lista->next->next = malloc(sizeof(struct element));
   lista->next->next->i = 3;
   lista->next->next->next = NULL;
   wyswietlListeBezGlowy(lista);
   zeruj(lista);
   wyswietlListeBezGlowy(lista);
   return 0;
}
```

```
struct element *polacz_listy(struct element *lista1, struct
element *lista2)
{
   //zal.: listy sa takiej samej dlugosci
   if (lista1 == NULL){
      return NULL;
   }
   // nowa lista
   struct element * new_list = lista1;
   // wskaznik do przechodzenia po nowej liscie
   struct element * wsk = new_list;
   // wskazniki do przechodzenia po listach 1 i 2
   struct element * wsk1 = lista1;
   struct element * wsk2 = lista2;
   // teraz petla
   while(wsk1 != NULL){
      wsk1 = wsk1->next;
      wsk->next = wsk2;
      wsk = wsk->next;
      wsk2 = wsk2->next;
      wsk->next = wsk1;
      wsk = wsk->next;
   }
   return new_list;
}

int main()
{
   struct element *lista1 = malloc(sizeof(struct element));
   lista1->i = 1;
   lista1->next = malloc(sizeof(struct element));
   lista1->next->i = 2;
   lista1->next->next = malloc(sizeof(struct element));
   lista1->next->next->i = 3;
   lista1->next->next->next = NULL;
   struct element *lista2 = malloc(sizeof(struct element));
   lista2->i = -5;
   lista2->next = malloc(sizeof(struct element));
   lista2->next->i = -6;
   lista2->next->next = malloc(sizeof(struct element));
   lista2->next->next->i = -7;
   lista2->next->next->next = NULL;
   wyswietlListeBezGlowy(lista1);
   wyswietlListeBezGlowy(lista2);
   struct element *lista3 = polacz_listy(lista1, lista2);
   wyswietlListeBezGlowy(lista3);
   // w tym zadaniu pierwotne listy nie sa juz
przechowywane nigdzie w pamieci w pierwotnej
kolejnosci
}
```

Usuniecie pierwszego el z listy bez głowy

```
struct element * usun(struct element * lista, int a)
{
   if (lista == NULL)
      return NULL;
   if(lista->i == a)
   {
      struct element * wsk = lista->next;
      free(lista);
      return wsk;
   }
   struct element * wsk= lista;
   while( (wsk->next!=NULL) && (wsk->next->i !=a) )
   {
      wsk=wsk->next;
   }
   if(wsk->next!=NULL)
   {
      struct element * wsk2 =wsk;
      wsk2=wsk2->next;
      wsk->next=wsk->next->next;

      free(wsk2);
   }
   return lista;
}
```

```
struct element* dodaj(struct element* lista,int a)
{
   struct element*wsk=malloc(sizeof(struct element));
   wsk->i=a;
   wsk->next=lista;
   return wsk;
}
```

Napisz funkcję dodajk o dwóch argumentach *Lista* typu *element i *a* typu *int* zwracającą wskaźnik do typu element. Funkcja powinna dodawać na koniec listy nowy element i zwracać wskaźnik do pierwszego elementu tak powiększonej listy.**

```
struct element * dodajk(struct element * Lista, int a)
{
   struct element * wsk;
   if(Lista == NULL)
   {
      Lista=wsk=malloc(sizeof(struct element));
   }
   else
   {
      wsk=Lista;
      while(wsk->next!=NULL)
      {
         wsk=wsk->next;
      }
      wsk->next=malloc(sizeof(struct element));
      wsk=wsk->next;
   }
   wsk->i=a;
   wsk->next=NULL;
   return Lista;
}
```

LISTY Z GŁOWĄ

Dodaj na koniec listy z głową

```
void dodajk(struct element * Lista, int a)
{
   struct element*wsk=Lista;
   while(wsk->next!=NULL)
   {
      wsk=wsk->next;
   }
   wsk->next=malloc(sizeof(struct element));
   wsk=wsk->next;
   wsk->i=a;
   wsk->next=NULL;
}
```

Usuwanie pierwszego el z listy z głową

```
void usun_pierwszy(struct elem** lista) {
    if (*lista == NULL) {
        printf("Lista jest pusta.\n");
        return;
    }
    struct elem* pierwszy = *lista;
    *lista = pierwszy->next;
    free(pierwszy);
}

int main() {
    // Tworzenie listy: 1 -> 2 -> 3
    printf("Przed usunięciem pierwszego elementu: ");
    struct elem* aktualny = lista;
    while (aktualny != NULL) {
        printf("%d ", aktualny->x);
        aktualny = aktualny->next;
    }
    usun_pierwszy(&lista);
    printf("\nPo usunięciu pierwszego elementu: ");
    aktualny = lista;
    while (aktualny != NULL) {
        printf("%d ", aktualny->x);
        aktualny = aktualny->next;
    }
    return 0;
}
```

Dodaj el na początek listy z głową

```
void dodaj(struct element* lista,int a)
{
    struct element*wsk=malloc(sizeof(struct
element));
    wsk->i=a;
    wsk->next=lista->next;
    lista->next=wsk;
}

int main()
{
    struct element *lista = malloc(sizeof(struct
element));
    lista->next = NULL;
    dodaj(lista, 5);
    dodaj(lista, 6);
    dodaj(lista, 7);
    wyswietlListeZGlowa(lista);
    return 0;
}
```

Porównanie list z głową

```
struct node {
    int x;
    struct node* next;
};
int porownaj_listy(struct node* lista1, struct node* lista2) {
    int licznik1 = 0;
    int licznik2 = 0;
    struct node* aktualny1 = lista1;
    while (aktualny1 != NULL) {
        if (aktualny1->x > 0) {
            licznik1++;
        }
        aktualny1 = aktualny1->next;
    }
    struct node* aktualny2 = lista2;
    while (aktualny2 != NULL) {
        if (aktualny2->x > 0) {
            licznik2++;
        }
        aktualny2 = aktualny2->next;
    }
    if (licznik1 == licznik2) {
        return 1;
    } else {
        return 0;
    }
}

int main() {
    // Tworzenie listy 1: 1 -> -2 -> 3
    struct node* lista1 = (struct node*)malloc(sizeof(struct node));
    lista1->x = 1;
    lista1->next = (struct node*)malloc(sizeof(struct node));
    lista1->next->x = -2;
    lista1->next->next = (struct node*)malloc(sizeof(struct node));
    lista1->next->next->x = 3;
    lista1->next->next->next = NULL;
    // Tworzenie listy 2: -1 -> 2 -> 4
    struct node* lista2 = (struct node*)malloc(sizeof(struct node));
    lista2->x = -1;
    lista2->next = (struct node*)malloc(sizeof(struct node));
    lista2->next->x = 2;
    lista2->next->next = (struct node*)malloc(sizeof(struct node));
    lista2->next->next->x = 4;
    lista2->next->next->next = NULL;
    int wynik = porownaj_listy(lista1, lista2);
    if (wynik == 1) {
        printf("Obie listy maja tyle samo dodatnich elementow.\n");
    } else {
        printf("Listy nie maja tyle samo dodatnich elementow.\n");
    }
    return 0;
}
```

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

struct element
{
   int i;
   struct element *next;
};

void zeruj(struct element *lista)
{
   struct element * wsk = lista->next;
   while(wsk != NULL)
   {
      wsk->i = 0;
      wsk = wsk->next;
   }
}

void wyswietlListeZGlowa(struct element *lista)
{
   if (lista->next == NULL)
   {
      printf("Lista jest pusta.\n---\n");
      return;
   }
   struct element *wsk = lista->next;
   while (wsk != NULL)
   {
      printf("%d\n", wsk->i);
      wsk = wsk->next;
   }
   printf("---\n");
}

int main()
{
   struct element *lista = malloc(sizeof(struct element));
   lista->next = malloc(sizeof(struct element));
   lista->next->i = 1;
   lista->next->next = malloc(sizeof(struct element));
   lista->next->next->i = -3;
   lista->next->next->next = malloc(sizeof(struct element));
   lista->next->next->next->i = 5;
   lista->next->next->next->next = NULL;
   wyswietlListeZGlowa(lista);
   zeruj(lista);
   wyswietlListeZGlowa(lista);
   return 0;
}
```

Napisz funkcję, która przyjmuje jako argument listę z głową o elementach typu:

```c
struct element {
  double x;
  struct element * next;
};
```

i zwraca sumę części całkowitych liczb znajdujących się na liście. W przypadku pustej listy, funkcja ma zwrócić zero. Stwórz jeden przypadek testowy.

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

struct element
{
   double x;
   struct element * next;
};

int suma(struct element * lista)
{
   int suma = 0;
   struct element * wsk = lista;
   while(wsk != NULL){
      suma += (wsk->x)/1;
      wsk = wsk->next;
   }
   return suma;
}

int main()
{
   struct element *lista = malloc(sizeof(struct element));
   lista->next = malloc(sizeof(struct element));
   lista->next->x = 1.2;
   lista->next->next = malloc(sizeof(struct element));
   lista->next->next->x = -3.5;
   lista->next->next->next = malloc(sizeof(struct element));
   lista->next->next->next->x = 5.7;
   lista->next->next->next->next = NULL;
   printf("%d\n", suma(lista));
   return 0;
}
```