

Ćwiczenie 5 z Podstaw programowania.

Język C++, programy pisane w
nieobiekowym stylu
programowania

Zofia Kruczkiewicz

Zakres

- Arytmetyka wskaźników, konwersja (rzutowanie) wskaźników
- Ćwiczenia z dostępu do dowolnego obszaru pamięci.

Do przygotowania na ćw. 6

- Funkcje przetwarzające teksty
 - omówienie standardowych funkcji (biblioteka `<string.h>`)
 - tworzenie własnych funkcji
- Dynamiczna alokacja pamięci – tablice jednowymiarowe o zmiennym rozmiarze - realokacja.

**Arytmetyka wskaźników, konwersja (rzutowanie)
wskaźników – wykonano równoważne definicje ciał
funkcji z cw4 za pomocą operacji na wskaźnikach**

Przykład 1: prototyp funkcji, wywołanie funkcji, definicje funkcji na kolejnym slajdzie

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

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

```
//prototypy funkcji, można pominąć nazwy argumentów
```

```
void suma ();
```

```
int suma1 (int a, int b);
```

```
void suma2 (int a, int b, int& suma);
```

```
void suma3 (int a, int b, int* suma);
```

```
int a=5, b=8, c;
```

```
void main()
```

```
{ int A=3, B=4, C=0;
```

```
printf("C: %d\n",C);
```

```
suma();
```

// wywołanie funkcji suma, używanie zmiennych globalnych a, b i c oraz lokalnej d

```
C=suma1(A, B);
```

// wywołanie funkcji suma1, zmienne A i B przekazane przez wartość,

```
printf("suma1: %d + %d = %d\n",A,B,C);
```

// pobranie wyniku funkcji suma1 do zmiennej C

```
C=0;
```

```
suma2(A,B,C);
```

//wywołanie funkcji suma2, zmienna C przekazana przez referencję

```
printf("suma2: %d + %d = %d\n",A,B,C);
```

// wartość zmiennej C uległa zmianie

```
C=0;
```

```
suma3(A,B,&C);
```

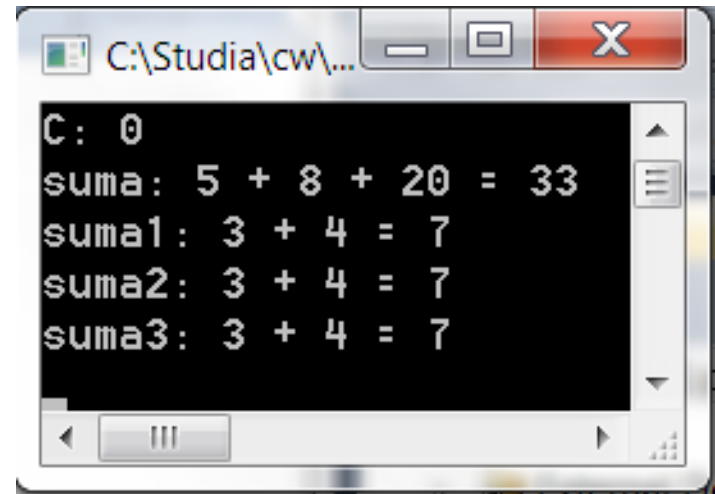
//wywołanie funkcji suma3, zmienna C przekazana przez wskaźnik

```
printf("suma3: %d + %d = %d\n ",A,B,C);
```

// wartość zmiennej C uległa zmianie

```
_getch();
```

```
}
```



```
C:\Studia\cw\...
C: 0
suma: 5 + 8 + 20 = 33
suma1: 3 + 4 = 7
suma2: 3 + 4 = 7
suma3: 3 + 4 = 7
```

Definicja funkcji 1' – operacje na wskaźnikach

//definicje funkcji

void suma()

```
{   int d=20;
    int* d1, *a1, *b1, *c1;
    d1=&d, a1=&a, b1=&b, c1=&c;
    *c1=*a1+*b1+*d1;           // a1, b1, c1 –wskaźniki do zmiennych globalnych; zmienna d – lokalna
    printf("suma: %d + %d + %d = %d\n",*a1,*b1,*d1,*c1);
}
```

int suma1 (**int** a, **int** b) //argumenty a, b – przekazywanie argumentów przez wartość

```
{   int* a1,*b1;
    a1=&a, b1=&b;
    return *a1 + *b1;           //zwracanie wyniku przez funkcję za pomocą instrukcji return
}
```

void suma2 (**int** a, **int** b, **int&** c) //argument c – przekazywanie argumentu przez referencję

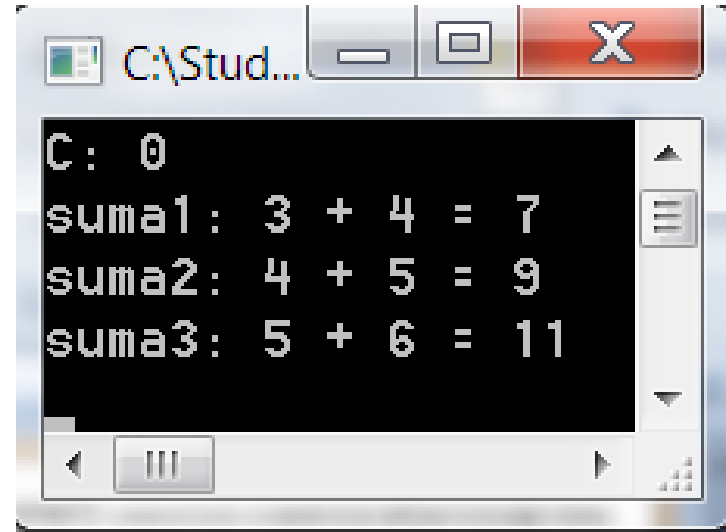
```
{   int* a1, *b1, *c1;           // gdzie zdefiniowano zmienną przekazaną przez referencję do c
    a1=&a, b1=&b, c1=&c;
    *c1= *a1+*b1;
}
```

void suma3 (**int** a, **int** b, **int*** c) //argument c – przekazywanie argumentu przez wskaźnik

```
{   int* a1, *b1, *c1;
    a1=&a, b1=&b, c1=c;
    *c1= *a1+ *b1;           // przypisanie sumy *a1+*b1 do miejsca w pamięci, gdzie zdefiniowano
}                               // zmienną przekazaną przez wskaźnik do c i dodatkowo wskazywaną przez c1
```

Przykład 2: prototyp funkcji, wywołanie funkcji, definicje funkcji uległy zmianie

```
#include <stdio.h>
#include <conio.h>
//prototypy funkcji
int suma1 (int a, int b);
void suma2 (int a, int b, int& suma);
void suma3 (int a, int b, int* suma);
void main()
{ int A=3, B=4, C=0;
  printf("C: %d\n",C);
  C=suma1(A, B);           //wywołanie funkcji, zmienne A i B przekazane przez wartość
    printf("suma1: %d + %d = %d\n",A,B,C);
  suma2(4,5,C);           //wywołanie funkcji, stałe 4 i 5 przekazane przez wartość
    printf("suma2: %d + %d = %d\n",4,5,C);
  suma3(5,6,&C);          //wywołanie funkcji, stałe 5 i 6 przekazane przez wartość
    printf("suma3: %d + %d = %d\n",5,6,C);
  _getch();
}
```



```
C: 0
suma1: 3 + 4 = 7
suma2: 4 + 5 = 9
suma3: 5 + 6 = 11
```

Definicja funkcji 2' – operacje na wskaźnikach

//definicje funkcji

```
int suma1 (int a, int b)
{   int* a1, *b1;
    a1=&a, b1=&b;
    return *a1+*b1;           // równoważna wyrażeniu return a+b;
}
```

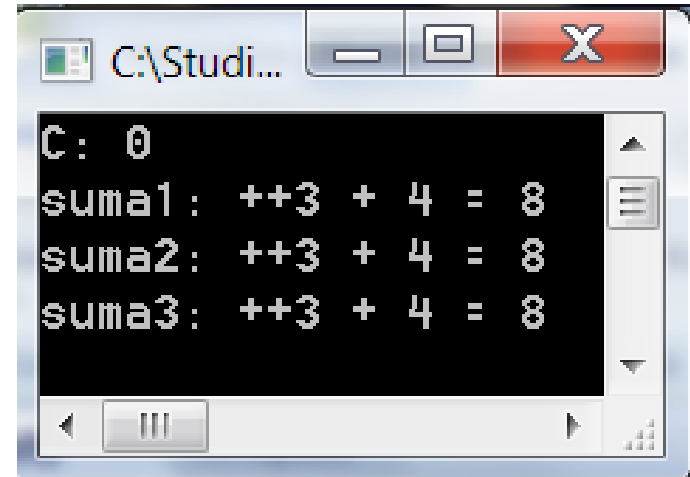
```
void suma2 (int a, int b, int& c)
{   int* a1, *b1, *c1;
    a1=&a, b1=&b, c1=&c;
    *c1=suma1(*a1,*b1);      //równoważna wyrażeniu c= a+b;
}
```

```
void suma3 (int a, int b, int* c)
{   int*a1, *b1,*c1;
    a1=&a, b1=&b, c1=c;
    *c1=suma1(*a1,*b1);     //równoważna wyrażeniu *c= a+b;
}
```

Przykład 3: prototyp funkcji, wywołanie funkcji, definicje funkcji zmienione

```
#include <stdio.h>
#include <conio.h>
//prototypy funkcji
int suma1 (int a, int b);
void suma2 (int a, int b, int& suma);
void suma3 (int a, int b, int* suma);

void main()
{ int A=3, B=4, C=0;
  printf("C: %d\n",C);
  C=suma1(A, B);           //wywołanie funkcji
  printf("suma1: ++%d + %d = %d\n",A,B,C);
  suma2(A,B,C);           //wywołanie funkcji
  printf("suma2: ++%d + %d = %d\n",A,B,C);
  suma3(A,B,&C);           //wywołanie funkcji
  printf("suma3: ++%d + %d = %d\n",A,B,C);
  _getch();
}
```



```
C:\Studi...
C: 0
suma1: ++3 + 4 = 8
suma2: ++3 + 4 = 8
suma3: ++3 + 4 = 8
```


Definicja funkcji 3'. Operacje na wskaźnikach – rzutowanie wskaźników

//definicje funkcji

```
int suma1 (int a, int b)
```

```
{ int* a1, *b1;
```

```
  a1=&a, b1=&b;
```

```
  return ++(*a1)+*b1;
```

```
}
```

//wartość a po zakończeniu funkcji znika

//wskazywane przez a1 i b1

```
void suma2 (int a, int b, int& c)
```

```
{ void*a1, *b1, * c1;
```

```
  a1=&a, b1=&b, c1=&c;
```

```
  *(int*)c1=suma1(*(int*)a1,*(int*)b1);
```

//równoważne wyrażeniu c= ++a+b; rzutowanie

//wskaźników typu void na typ int

```
  *(int*)a1=3;
```

// **ostrożnie ze zmianą typu zmiennej!!!**

```
  *(int*)b1=10;
```

//wartości a i b po zakończeniu funkcji znikają

```
}
```

//wskazywane przez a1 i b1

```
void suma3 (int a, int b, int* c)
```

```
{ short* a1, * b1, * c1;
```

```
  a1=(short*)&a, b1=(short*)&b, c1=(short*)c;
```

//rzutowanie wskaźników typu int na typ short

```
  *(int*)c1=suma1(*(int*)a1,*(int*)b1);
```

// równoważne wyrażeniu *c= ++a+b; rzutowanie

//wskaźników typu short na typ int

```
  *(int*)a1=9;
```

//ostrożnie ze zmianą typu zmiennej!!!

```
  *(int*)b1=11;
```

//wartości a i b po zakończeniu funkcji znikają

```
}
```

//wskazywane przez a1 i b1

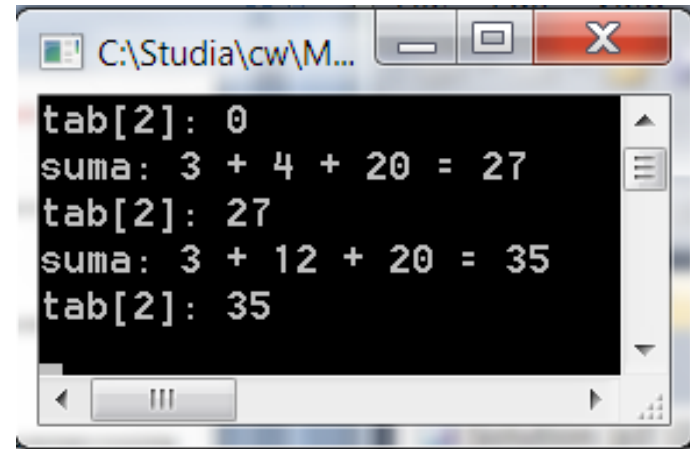
Przykład 4: prototyp funkcji, wywołanie funkcji, przekazywanie tablicy do funkcji

```
#include <stdio.h>
#include <conio.h>
//prototyp funkcji
void suma (int tab[]);
```

```
void main()
{ int tab []={3, 4, 0};
  printf("tab[2]: %d\n", tab[2]);
  suma(tab);
  printf("tab[2]: %d\n", tab[2]);
  suma(tab);
  printf("tab[2]: %d\n", tab[2]);
  _getch();
}
```

//definicja funkcji

```
void suma(int tab[])
{ int d=20;
  tab[2]=tab[0]+tab[1]+d;
  printf("suma: %d + %d + %d = %d\n", tab[0], tab[1], d, tab[2]);
  tab[1]=12;
}
```



```
tab[2]: 0
suma: 3 + 4 + 20 = 27
tab[2]: 27
suma: 3 + 12 + 20 = 35
tab[2]: 35
```

//wywołanie funkcji suma, zmienna tab przekazana przez domyślny
// wskaźnik; wartości elementów tab[1] i tab[2] uległa zmianie
//wywołanie funkcji suma, zmienna tab przekazana przez domyślny
// wskaźnik; wartość elementu tab[2] uległa zmianie

Przykład 4': prototyp funkcji, wywołanie funkcji, przekazywanie tablicy do funkcji

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

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

```
void suma (int *);
```

```
void main()
```

```
{ int tab []={3, 4, 0};
```

```
    printf("tab[2]: %d\n", *(tab+2));
```

```
    suma(tab);
```

```
    //wywołanie funkcji suma, zmienna tab przekazana przez domyślny
```

```
    printf("tab[2]: %d\n", *(tab+2));
```

```
    // wskaźnik; wartości elementów tab[1] i tab[2] uległa zmianie
```

```
    suma(tab);
```

```
    //wywołanie funkcji suma, zmienna tab przekazana przez domyślny
```

```
    printf("tab[2]: %d\n", *(tab+2));
```

```
    // wskaźnik; wartość elementu tab[2] uległa zmianie
```

```
    _getch();
```

```
}
```

//definicja funkcji – zastosowanie operacji na wskaźnikach (rzutowanie wskaźników)

```
void suma(int *tab)
```

```
{ long d=20;
```

```
int* d1, *tab1;
```

```
d1=(int*)&d, tab1=tab; // rzutowanie na int
```

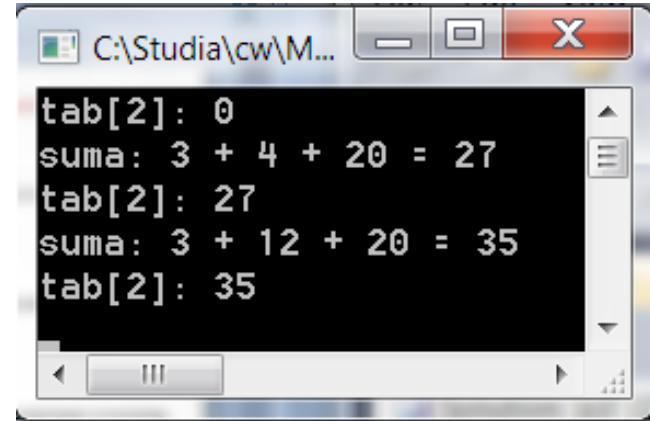
```
*(tab1+2)=*(tab1+0)+*(tab1+1)+*d1; //obliczenia wykonywane są na wartościach typu int, a nie typu long
```

```
    //dzięki zamianie typu long na int (zmienna wskaźnikowa d1)
```

```
printf("suma: %d + %d + %d = %d\n", *(tab1+0), *(tab1+1), *d1, *(tab1+2));
```

```
*(tab1+1)=12;
```

```
}
```



```
C:\Studia\cw\M...
tab[2]: 0
suma: 3 + 4 + 20 = 27
tab[2]: 27
suma: 3 + 12 + 20 = 35
tab[2]: 35
```

**Zadania do wykonania – w definicji ciała każdej funkcji
należy zastosować operacje na wskaźnikach wg
przykładów 1-4**

Zad 1 (obowiązkowy) – w definicji ciała każdej funkcji należy użyć wskaźników wg przykładów 1-4

1. Napisz program, który w opcjach wywołuje funkcje, które wykonują następujące czynności na tablicy zdefiniowanej jako `int tab[N]`:
 - 1.1. dodaje po jednym elemencie typu `int` do tablicy na pozycję o indeksie `ile`. Podczas wstawiania pierwszego elementu `ile` powinno być równe 0; po wstawieniu należy powiększyć wartość `ile` o 1. Po ponownym wywołaniu funkcji kolejny element należy wstawiać jako element tablicy o indeksie `ile`, gdy `ile` spełnia warunek: `ile < N`. Po wykonaniu tej opcji, jeśli element został wstawiony do tablicy, należy zawsze powiększyć `ile` o jeden. Zmienną `ile` należy przekazywać przez referencję. Tablica powinna być przekazana przez domyślny wskaźnik.
 - 1.2. wyszukuje element o zadanej wartości z klawiatury i zwraca jego wartość oraz wartość indeksu. Indeks znalezionego elementu należy przekazać przez wskaźnik, a wartość znalezionego elementu należy przekazać przez wynik funkcji. Tablica powinna być przekazana przez domyślny wskaźnik.
 - 1.3. wyświetla wartość przekazanego elementu tablicy oraz wartość jego indeksu- zastosować przekazanie przez wartość tych danych. Propozycja nagłówka funkcji:
prototyp: `void wyswietl_el(int element, int indeks);`
wywołanie: `wyswietl_el(tab[indeks], indeks);` // wartość zmiennej `indeks` wprowadzona z klawiatury. Należy zbadać, czy spełnia warunek `0 <= indeks < ile`
 - 1.4. wyświetla zawartość tablicy, jeśli zawiera dane. Należy przekazać tablicę przez domyślny wskaźnik i wartość `ile` przez wartość.

Uwagi:

- Należy wprowadzić zmienną `ile`, która przechowuje liczbę elementów tablicy. Wartość `ile` równa zero świadczy o tym, że tablica jest pusta. Wartość różna od zera oznacza, że w tablicy są dane. Nie może ona przekroczyć wartości `N` - `ile=N` oznacza, że tablica jest pełna. Wartość `N` musi być zadana programie, np. **`const int N=10;`**
- Należy do każdej funkcji dodać komentarz wyjaśniający znaczenie wybranego sposobu przekazania zmiennych. Należy również dodać w każdej funkcji komentarz wyjaśniający sposób działania kodu reprezentującego algorytmy z p. 1.1 – 1.4:
 - 1.1. sposób dodawania elementów do tablicy
 - 1.2 . Sposób wyszukiwania elementów w tablicy
 - 1.3. sposób wyświetlania elementów tablicy

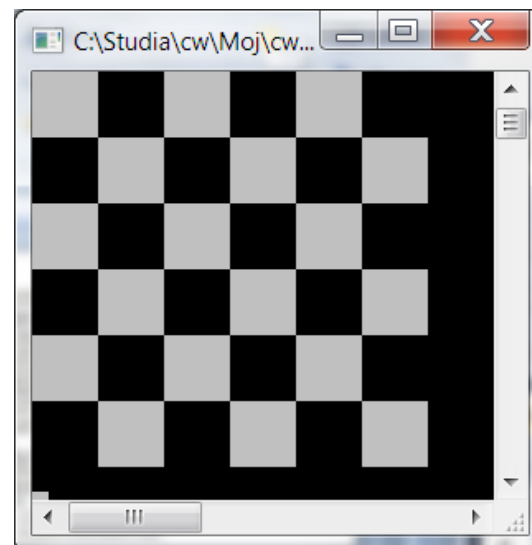
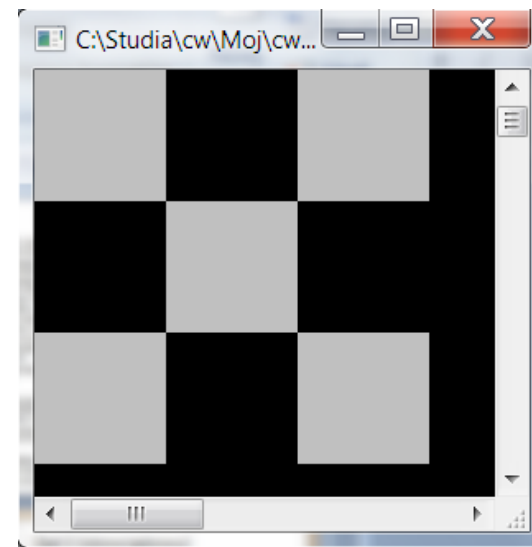
Przykład rozwiązania zad1 – należy zdefiniować funkcje przetwarzające zawartość tablicy jednowymiarowej elementów typu int.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
//prototypy 4 funkcji

void main()
{ int opcja; // zmienna wyboru typu int
// tutaj należy zdefiniowac tablice, zmienną ile, stałą N itd...
do
{ system("cls");
printf("Jesli naciśniesz klawisz:\n");
printf("1 - Dodawanie kolejnego elementu do tablicy\n");
printf("2 - Wyszukiwanie zadanego elementu w tablicy\n");
printf("3 - Wyświetlenie elementu o podanym indeksie 3\n");
printf("4 - Wyświetlenie wszystkich elementów wstawionych do tablicy\n");
printf("0 - Koniec programu\n");
scanf("%d",&opcja);
switch(opcja)
{ case 1 :printf("Naciśniesz klawisz 1 - operacja 1.1\n"); break; //wstawić wywołanie funkcji wstawiającej kolejny element do tablicy wg 1.1.
case 2 :printf("Naciśniesz klawisz 2 - operacja 1.2\n"); break;// wstawić wywołanie funkcji wyszukującej element w tablicy wg p.1.2
case 3 :printf("Naciśniesz klawisz 3 - operacja 1.3\n"); break;// wstawić wywołanie funkcji wg 1.3 wyświetlającej element tablicy o zadanym indeksie
case 4 :printf("Naciśniesz klawisz 4 - operacja 1.4\n"); break;// wstawić wywołanie funkcji wyświetlającej zawartość tablicy wg p.1.4
case 0 :printf("Naciśniesz klawisz 0 - koniec programu\n");break;
default :printf("Naciśniesz niewłaściwy klawisz\n");
}
_getch(); //wstrzymanie programu przed clrscr()przez dodatkowe naciśnięcie klawisza
}while (opcja!=0);
}
//definicje 4 funkcji
```

Zad 2 (obowiązkowy) - w definicji ciała każdej funkcji należy użyć wskaźniki wg przykładów 1-4

- Napisz program, który zawiera funkcję rysującą szachownicę na ekranie. Przez nagłówek funkcji należy przekazać przez wartość informację o rozmiarze boku pola szachownicy (kwadrat). Algorytm powinien zawierać dwie pętle – jedna zagnieżdżona w drugiej. Nie należy stosować tablicy. Należy zastosować operator warunkowy `?:` zamiast instrukcji `if else` przy rozpoznawaniu, jakie pole szachownicy należy rysować. Rysowanie należy wykonać za pomocą funkcji `printf`. Można wybrać rodzaj znaku do rysowania pól szachownicy (wartość zadana w kodzie programu).
- Funkcję należy wywoływać dowolną liczbę razy, przekazując podaną z klawiatury długość boku pola szachownicy. Zmiana wielkości boku powoduje rysowanie różnej liczby pól szachownicy, ponieważ program powinien zachować ten sam rozmiar boku całej szachownicy. Po naciśnięciu klawisza 'k' program powinien zakończyć się.
- Dodać komentarze w kodzie programu wyjaśniające, jak on reprezentuje algorytm rysowania figury.



Przykład rozwiązania zad2 – należy zdefiniować funkcję rysującą szachownicę za pomocą funkcji printf

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
//prototyp funkcji
void szachownica(int bok);

void main()
{ char klucz;
  int bok;
do
{
  system("cls");
  printf("Wprowadz dlugosc boku: ");
  scanf("%d",&bok);
  szachownica(bok);
  printf("Czy koniec programu: k - koniec, dowolny inny klawisz - dalej\n");
  klucz=_getch();
}while(klucz!='k');
}
//definicja funkcji
void szachownica(int bok)
{ //tutaj należy napisac kod funkcji rysujacej szachownice
}
```

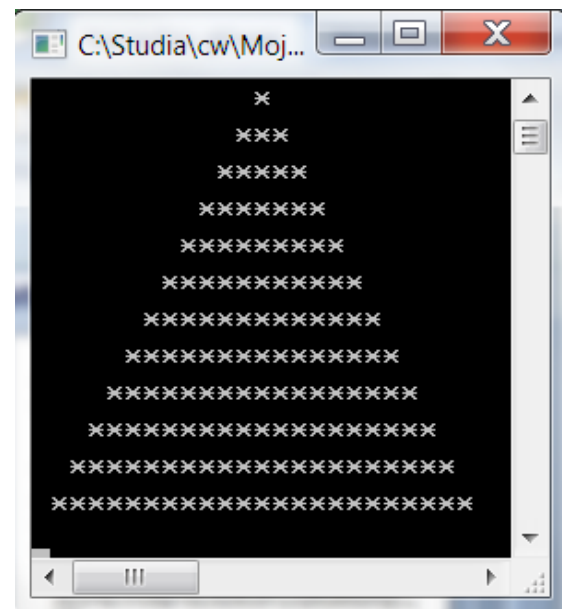
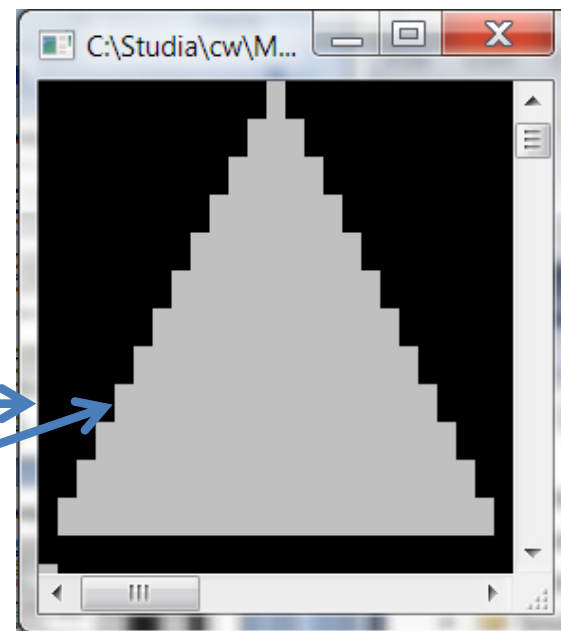

Zad 3 (obowiązkowy) - w definicji ciała każdej funkcji należy użyć wskaźniki wg przykładów 1-4

- Napisz program, który zawiera funkcję rysującą „choinkę” na ekranie. Przez nagłówek funkcji należy przekazać przez wartość informację o wysokości choinki (trójkąt równoramienny) oraz rodzaj znaku do wypełnienia powierzchni trójkąta. Algorytm powinien zawierać jedną pętlę główną, w której rysowany kolejny wiersz figury. Każdy wiersz figury rysowany jest za pomocą dwóch pętli (zagnieżdżonych w pętli głównej) o różnej liczbie wykonań w kolejnym wierszu.

Pierwsza pętla rysuje tło (wypisywane spacje), a druga pętla wypisuje znaki reprezentujące kolejny wiersz trójkąta.

Nie należy stosować tablicy. Rysowanie należy wykonać za pomocą funkcji **printf**.

- Funkcję należy wywoływać dowolną liczbę razy, przekazując przez wartość podane z klawiatury wysokość choinki i rodzaj znaku wypełniającego powierzchnię figury. Po naciśnięciu klawisza 'k' program powinien zakończyć się.
- Dodać komentarze w kodzie programu wyjaśniające, jak on reprezentuje algorytm rysowania figury.
- Budowa programu jest podobna do budowy programu z zad2.



Zad 4 (obowiązkowy) - w definicji ciągu każdej funkcji należy użyć wskaźniki wg przykładów 1-4

1. Napisz program, który zawiera funkcję, która bada, czy wprowadzony łańcuch znaków jest palindromem tekstowym. Łańcuch znaków należy przekazać do funkcji przez listę argumentów. Jeśli wprowadzony łańcuch jest palindromem, funkcja przez wynik zwraca wartość 1, a w przeciwnym wypadku zwraca wartość 0.
2. Funkcję należy wywoływać dowolną liczbę razy. Po wywołaniu funkcji należy zbadać zwrócony przez nią wynik i wyświetlić na ekranie słowo „tak” (wynik równy 1) lub „nie” (wynik równy 0). Po naciśnięciu klawisza ‘k’ program powinien zakończyć się. Dodać komentarze w kodzie programu wyjaśniające, jak on reprezentuje działanie algorytmu badania łańcucha znaków w celu rozpoznania palindromu.

Przykłady palindromów tekstowych

Wprowadzony ciąg znaków: **maloolam**

Tablica znaków

Indeks tablicy	0	1	2	3	4	5	6	7
Wartość elementu	m	a	l	o	o	l	a	m

Wprowadzony ciąg znaków: **malotolam**

Tablica znaków

Indeks tablicy	0	1	2	3	4	5	6	7	8
Wartość elementu	m	a	l	o	t	o	l	a	m

Słowo jest palindromem jest wtedy, gdy wartość znaku pierwszego jest równa ostatniemu, i drugi przedostatniemu itd.

W przypadku pierwszej tablicy spełnione są powyższe warunki, ponieważ wyraz jest palindromem

tab[0] == tab[7]

tab[1] == tab[6]

tab[2] == tab[5]

tab[3] == tab[4]

W przypadku drugiej tablicy spełnione są powyższe warunki, ponieważ wyraz jest palindromem :

tab[0] == tab[8]

tab[1] == tab[7]

tab[2] == tab[6]

tab[3] == tab[5]

Przy nieparzystej liczbie elementów pomija się sprawdzanie elementu środkowego.

Przykład rozwiązania zad4 – należy zdefiniować funkcję palindrom

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

int palindrom(char wiersz[]);
void main()
{  const int N=30;
   char klucz;
   char wiersz[N];
   int wynik;
do
{ system("cls");
  printf("Wprowadz lancuch, nie dluzszy niz %d znakow: ",N);
  scanf("%29s",wiersz);
  wynik=palindrom(wiersz);
  printf("Czy to jest palindrom?\n");
  if (wynik==1)
    printf("tak\n");
  else
    printf("nie\n");
  printf("Czy koniec programu: k - koniec, dowolny inny klawisz - dalej\n");
  klucz=_getch();
}while(klucz!='k');
}
int palindrom(char wiersz[])
{ //tutaj należy napisac kod funkcji badającej zawartosc tablicy wiersz
  // funkcja zwraca warunkowo 1, gdy wykryto palindrom, 0, gdy nie wykryto
return 1;
}
```

Zad5 (dodatkowe) - należy zastosować funkcje oraz umieścić komentarze w programie, określające który krok algorytmu reprezentuje komentowany kod. **W definicji ciała funkcji należy użyć wskaźniki wg przykładów 1-4**

Algorytm wyszukania liczb pierwszych metodą sita Eratostenesa. Należy wyznaczyć wszystkie liczby pierwsze w podziorze liczb naturalnych $\{1..N\}$ za pomocą algorytmu sita Eratostenesa wg podanego algorytmu. Należy wykonać schemat blokowy i program z użyciem tablicy ***int tab[N]***.

- 1) Utworzyć tablicę zawierającą N elementów i wstawić do każdego elementu wartość 0
 - 1.1) Podaj ***N*** z klawiatury
 - 1.2) Jeśli ***N < 2***, powtórz krok 1.1
 - 1.3) Ustaw ***i := 2***
 - 1.4) Dopóki *i ≤ N* wykonuj kolejne kroki, w przeciwnym wypadku przejdź do kroku 2**
 - 1.4.1) wstaw 0 do elementu tablicy o indeksie *i*
 - 1.4.2) zwiększ ***i := i + 1*** i przejdź do **kroku 1.4.**
- 2) Zakłada się, że pewne indeksy elementów są szukanymi liczbami pierwszymi i po zakończeniu algorytmu elementy tablicy o tych indeksach będą zawierać wartość 0, natomiast pozostałe elementy mają wartość 1, ponieważ nie są liczbami pierwszymi. Stąd należy wstawić na początku wartość 1 do elementu o indeksie równym 1, ponieważ 1 nie jest liczbą pierwszą.
- 3) Ustawić ***ost_Liczp := 1;***

- 4) Na podstawie faktu, że każda liczba złożona nie większa niż N ma dzielnik nie większy niż \sqrt{N} , wykonuj kolejne kroki, gdy $ost_Liczp * ost_Liczp \leq N$, w przeciwnym wypadku przejdź do kroku 5:
- 4.1) Należy zwiększyć ost_Liczp o 1: $ost_Liczp := ost_Liczp + 1$
- 4.2) Wykonuj kolejne kroki, jeśli jest prawdziwy warunek $ost_Liczp \leq N$ and $tab[ost_Liczp] = 1$, w przeciwnym wypadku przejdź do kroku 4.3.
- 4.2.1) zwiększaj ost_Liczp o 1: $ost_Liczp := ost_Liczp + 1$ (poszukiwanie kolejnej liczby pierwszej, czyli elementu tablicy o indeksie ost_Liczp nie zawierającej wartości 1)
- 4.4.2) przejdź do kroku 4.2
- 4.3) Należy wyznaczyć podwojoną wartość ost_Liczp ost i wyznaczyć numer i kolejnej liczby, która nie jest liczbą pierwszą: $i := ost_Liczp * 2$ (rozpoczęcie kolejnego etapu wykreślenia liczb, które nie są liczbami pierwszymi)
- 4.4) Dopóki $i \leq N$, wykonaj w kolejnych krokach eliminacje liczb, które nie są liczbami pierwszymi, ponieważ są ich wielokrotnościami, w przeciwnym wypadku przejdź do kroku 4.
- 4.4.1) wstaw wartość 1 to elementu tablicy o wierszu równym i : $tab[i] := 1$
- 4.4.2) dodaj wartość ost_Liczp do i : $i := i + ost_Liczp$, następnie przejdź do kroku 4.4
- 5) Wyświetl zawartość tablicy na ekranie:
- 5.1) wstaw $i := 1$
- 5.2) dopóki $i \leq N$ wykonuj kolejne kroki, w przeciwnym wypadku zakończ algorytm
- 5.2.1) jeśli $tab[i] = 0$, wyświetl indeks elementu jako wartość kolejnej liczby pierwszej
- 5.2.2) wyznacz kolejny indeks $i := i + 1$ i przejdź do kroku 5.2.

Wyjaśnienie:

Wyrażenie: $ost_Liczp := ost_Liczp + 1$ oznacza, że zmiennej ost_Liczp została przypisana wartość o 1 większa od poprzedniej