Programowanie strukturalne – podstawowe typy danych, instrukcje, struktury danych i funkcje w języku C

Opracowali:	Sławomir Samolej, Andrzej Bożek	Aktualizacja:	Dawid Warchoł
	Politechnika Rzeszowska		-
	Katedra Informatyki i Automatyki		-

1. Wprowadzenie

Programowanie strukturalne na podstawowym poziome polega na umiejętności zastosowania głównych typów danych, struktur danych, instrukcji języka i zasad dekomponowania programu na podprogramy w budowaniu programów realizujących pewną podstawową grupę algorytmów. Materiał zawarty w ćwiczeniu dotyczy wybranych podstawowych składników języka C i umiejętności ich podstawowego zastosowania.

1.1 Podstawowe typy danych

W języku C zdefiniowano cztery podstawowe typy danych:

char mała liczba całkowita, zwykle służąca do przechowywania zakodowanych w standardzie ASCII znaków;

int liczba całkowita podstawowej długości;

float liczba zmiennoprzecinkowa podstawowej precyzji;

double liczba zmiennoprzecinkowa podwójnej precyzji.

1.2 Tworzenie zmiennych

Instrukcja utworzenia zmiennej zawiera najpierw określenie typu zmiennej, następnie jej nazwy.

Przykład:

```
{
	int a;
	float b;
}
```

W przykładowym fragmencie programu utworzono dwie zmienne `a` i `b`. Zmienna `a` jest typu `int`, a zmienna `b` typu `float`.

Uwaga 1: Nazwa zmiennej może składać się z liter, cyfr i znaku podkreślenia `_`. Nie może jednak zaczynać się od cyfry.

Uwaga 2: Należy zwrócić uwagę, że po każdej instrukcji piszemy średnik (znak `;`).

1.3 Nadawanie wartości zmiennym

Instrukcja nadawania wartości zmiennej polega na podaniu nazwy zmiennej i po znaku równości podaniu wartości liczbowej przypisanej tej zmiennej.

Przykład:
```
{
	int a;
	a=5;
}
```

Zmiennej a nadano wartość 5.

Można to również zapisać skrótowo, w jednej linijce:
```
{
	int a = 5;
}
```

1.4 Operacje wejścia-wyjścia

Podstawowe operacje wejścia-wyjścia umożliwiają pobieranie oraz wypisywanie danych tekstowych i liczbowych z konsoli. Poniżej zostaną podane najprostsze metody odwoływania się do wejścia-wyjścia. Pełna dokumentacja i omówienie znajdują się np. na stronie:
https://cpp0x.pl/dokumentacja/standard-C/cstdio-stdio-h/492

Uwaga:
Aby można było skorzystać ze standardowych mechanizmów wejścia-wyjścia w języku C, należy na początku treści programu wskazać plik biblioteki **stdio** zawierającej odpowiednie nagłówki funkcji wej/wyj:
```
#include <stdio.h>
```

Do wyprowadzania danych na konsolę służą funkcje **printf** oraz **putchar**.

Funkcję **printf** w podstawowej konfiguracji można zastosować do wypisywania pojedynczych wartości liczbowych.

Przykład:
```
{
	int a;
	a=5;
	printf("%i",a);
}
```

Utworzonej zmiennej całkowitej a nadano wartość 5, a następnie wypisano wartość tej zmiennej na ekranie w postaci dziesiątkowej. Funkcja **printf** najpierw jest informowana w jaki sposób ma wyświetlić daną zmienną (ciąg odpowiednich znaków w cudzysłowie), a następnie, zawartość której zmiennej ma być wypisana na ekranie (nawa zmiennej po przecinku).
Uwaga:
Wybrane kombinacje znaków, pozwalające na wyświetlenie zawartości zmiennych w różnych formatach:
`%i` – zmienna całkowita w postaci dziesiątkowej;
`%x` – zmienna całkowita w postaci szesnastkowej;
`%f` - zmienna zmiennoprzecinkowa w postaci dziesiątkowej.

Funkcję **printf** można również stosować do wypisywania tekstu.

Przykład:
```
{
	printf("To jest tekst");
}
```

Na ekranie pojawi się tekst: **To jest tekst**.

Za pomocą **printf** można również wypisywać tekst razem z wartościami zmiennych.

Przykład:
```
{
	int a;
	a = 5;
	printf("Zmienna a jest równa: %i.",a);
}
```

Na ekranie pojawi się tekst: **Zmienna a jest równa 5**.

Funkcja **putchar** w podstawowym zastosowaniu służy do wyprowadzania wartości jednej zmiennej w postaci zdekodowanej na znak ASCII.

Przykład:
```
#include <stdio.h>

int main()
{
	char c;
	c='a';
	putchar(c);
	putchar('\n');
	return 0;
}
```
Utworzonej zmiennej c nadano wartość liczbową odwzorowującą numer w tablicy ASCII przypisany literze a. Zawartość zmiennej w postaci znaku została wypisana na konsoli. Dodatkowo na konsoli wypisano znak specjalny `\n`, który powoduje przejście do nowej linii. Należy zwrócić uwagę, że program rozpoczyna się od nagłówka `int main()`, a ostatnią instrukcją jest `return 0`. Więcej na ten temat można przeczytać w podrozdziale 1.12.

Do podstawowego pobierania danych z konsoli służą funkcje **scanf** oraz **getchar**. Funkcja **scanf** w podstawowym trybie swojej pracy może służyć do pobierania jednej wartości liczbowej z konsoli.

Przykład:
```
{
	float b;
	scanf("%f",&b);
}
```

Do utworzonej zmiennej zmiennoprzecinkowej `b` wprowadzona zostanie wartość z konsoli.

Uwaga: Aby wprowadzenie odbyło się poprawnie, przed nazwą zmiennej w wywołaniu funkcji **scanf** należy podać znak `&`.

Funkcja **getchar** w podstawowym trybie swojej pracy może służyć do pobierania jednego znaku z konsoli.

Przykład:

```
{
      char znak;
      znak = getchar();
}
```

Do zmiennej znak zostanie wprowadzony kod litery podanej na konsoli.

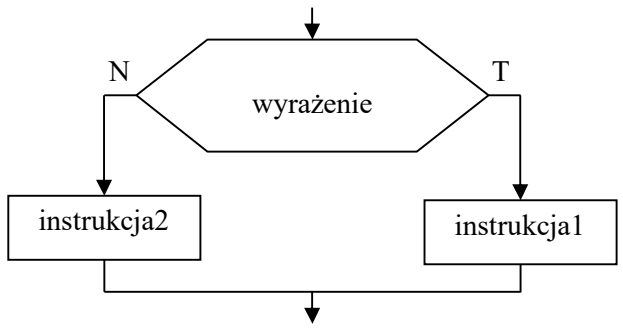
1.5 Operatory porównania

Do porównywania wartości liczbowych w języku C służą operatory porównania:

Operator	Operacja
`a == b`	Czy a jest równe b?
`a != b`	Czy a jest różne od b?
`a > b`	Czy a jest większe od b?
`a < b`	Czy a jest mniejsze od b?
`a >= b`	Czy a jest większe lub równe od b?
`a <=b`	Czy a jest mniejsze lub równe od b?

1.6 Instrukcja warunkowa - if

Jedną z podstawowych instrukcji w języku C jest instrukcja warunkowa, która pozwala na rozgałęzienia przebiegu działania programu (jeśli spełniony jest pewien warunek, to wykonaj działanie A, jeśli nie to wykonaj działanie B). Schemat blokowy i składnia instrukcji ma postać:




```
if(wyrażenie)
{
      instrukcja1;
}
else
{
      instrukcja2;
}
```

Jeśli wartość wyrażenia jest uznawana za prawdziwą, to wykonana zostanie instrukcja1, w przeciwnym wypadku – instrukcja2.

Przykład:

```
{
      int a=4, b=5;
      if (a > b) //jeśli a > b
      {
            printf("A jest wieksze od B");// to pisz: A jet wieksze od B
            b = 3; // b nadaj wartość 3
      }
```

```
        else
        {
                printf("A jest niewieksze od B"); //w przeciwnym wypadku pisz:
                                                  // A jest niewieksze od B
        }
}
```

Uwaga: Tekst w danej linii programu po serii znaków // jest traktowany jako komentarz. Pomiędzy nawiasami klamrowymi można wprowadzić dowolną ilość instrukcji i wszystkie się wykonają albo w wariancie „if” albo „else”.

1.7 Operatory arytmetyczne

Operacje arytmetyczne można wykonywać przy pomocy następujących *operatorów*:

Operator	Operacja
-	Odejmowanie
+	Dodawanie
*	Mnożenie
/	Dzielenie
%	Reszta z dzielenia (tylko dla zmiennych całkowitych)

Przykład:

```
{
        int a, b, c;
        a=5;
        b=6;
        a = a+1; //zwiększenie a o 1
        printf("%i",a); putchar('\n');
        c = b%6; //reszta z dzielenia b przez 6
        printf("%i",c);
}
```

1.8 Operatory logiczne

Do budowania bardziej złożonych wyrażeń logicznych w języku C można posłużyć się operatorami logicznymi:

Operator	Operacja
&&	Logiczne „i”
||	Logiczne „lub”
!	Logiczne „nieprawda, że”

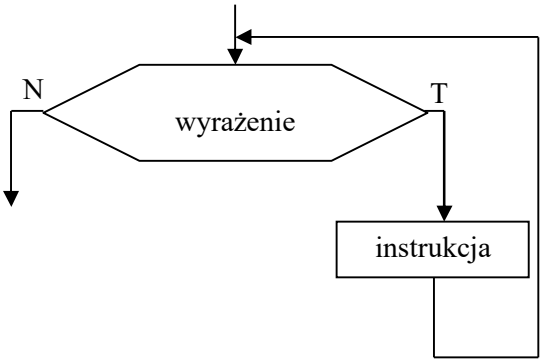
Przykład:

```
{
        int a;
        scanf("%i",&a);
        if (a > 6 && a < 10) //jeśli a > 6 i a < 10
                printf("warość a mieści się w przedziale (6,10)");
}
```

Tworzona jest zmienna a. Przypisuje się jej wartość całkowitą wprowadzoną z konsoli. Dokonuje się sprawdzenia, czy zmienna mieści się w zadanym przedziale.

1.9 Pętla while

Do wykonywania cyklicznych obliczeń w języku C stosuje się instrukcje cyklu (pętli). Podstawową instrukcją cyklu w języku C jest instrukcja „while”. Schemat blokowy i składnia instrukcji ma postać:




```
while(wyrażenie)
{
        instrukcja;
}
```

Jeśli wartość wyrażenia zostanie uznana za prawdziwą, to wykonana będzie instrukcja i program ponownie przejdzie do sprawdzenia wyrażenia. Instrukcja będzie powtarzana dotąd, aż wyrażenie przestanie być prawdziwe z punktu widzenia języka C.

Przykład:

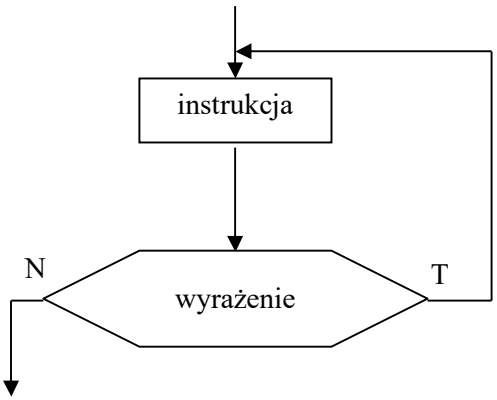
```
{
        int i = 10;
        while(i<10)
        {
                printf("%i",i);
                putchar('\n');
                i=i+1;
        }
}
```

Tworzona jest zmienna całkowita `i`. Nadawana jest jej wartość 0. Dopóki wartość przechowywana w zmiennej `i` jest mniejsza od 10:

- wartość `i` jest wypisywana na konsoli;
- wartość `i` jest zwiększana o 1.

1.10 Pętla do-while

Język C oferuje kilka instrukcji cyklu (pętli). Oprócz omówionej wcześniej instrukcji „while”, stosować można instrukcje „do while” i „for”. Schemat blokowy i składnia instrukcji „do while” ma postać:




```
do
{
        instrukcja;
}
while(wyrażenie);
```

Instrukcja będzie wykonywana tak długo, dopóki spełniony będzie warunek zawarty w wyrażeniu „while". Warto zauważyć, że najpierw wykonywana jest instrukcja, a następnie sprawdzana wartość wyrażenia. Oznacza to, że pętla „do while" wykonuje się przynajmniej raz, w przeciwieństwie do pętli „while" i „for", które mogą nie wykonać się ani razu. Należy pamiętać, że na końcu warunku w pętli „do while" wstawiamy średnik.

Przykład:

```
{
	int i = 0;
	do
	{
		printf("%i",i);
		putchar('\n');
		i=i+1;
	}
	while(i<10);
}
```

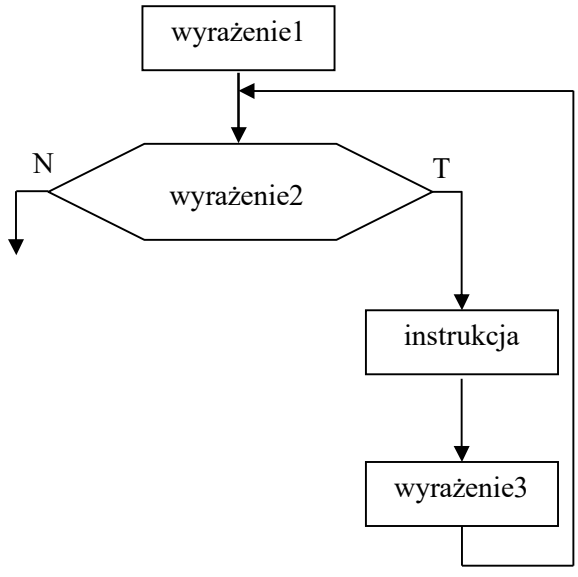
Tworzona jest zmienna całkowita `i`. Nadawana jest jej wartość 0. W pętli:

- wartość `i` jest wypisywana na konsoli;
- wartość `i` jest zwiększana o 1

dopóki wartość przechowywana w zmiennej `i` jest mniejsza od 10:

1.11 Instrukcja for

Praktyka programistyczna wskazuje, że często spotyka się pewien ciąg instrukcji, który można opisać za pomocą następującego schematu blokowego:




```
for(wyrażenie1;wyrażenie2;wyrażenie3)
{
	instrukcja;
}
```

Powyższy schemat blokowy można łatwo zapisać przy pomocy pętli „while”:

```
wyrażenie1;
while(wyrażenie2)
{
   instrukcja;
   wyrażenie3;
}
```

Z uwagi na częste stosowanie opisanego ciągu operacji, w języku C zaproponowano zastąpienie go osobną instrukcją pętli „for”:

```
for(wyrażenie1; wyrażenie2; wyrażenie3)
{
   instrukcja;
}
```

Przykład:

```
{
      int i;
      for(i=0; i<10; i=i+1)
      {
            printf("%i",a);
            putchar('\n');
      }
}
```

W nowszych standardach języka C dopuszczalne jest deklarowanie wewnątrz nagłówka pętli for zmiennej, która jest wykorzystywana jako licznik. W związku z tym oba poniższe fragmenty kodu są poprawne:

Przykład 1:

```
{
      int i;
      for(i=0; i<10; i=i+1)
}
```

Przykład 2:

```
{
      for(int i=0; i<10; i=i+1)
}
```

Instrukcje zwiększania wartości o 1 (inkrementacji) i zmniejszania wartości zmiennej o 1 (dekrementacji) można zapisać w trzech różnych formach. Pierwsze z nich są formami standardowymi, a dwie kolejne są formami skróconymi.

Inkrementacja:

```
i = i + 1;
i += 1;
i++;
```

Dekrementacja:

```
i = i - 1;
i -= 1;
i--;
```

1.12 Tablice jednowymiarowe zawierające liczby

Tablica jest ciągiem elementów tego samego typu, znajdujących się jeden za drugim w pamięci. Podstawowe informacje o tablicy to jej długość i typ danych jej elementów, np.:

```
int tab1[30]; //30-elementowa tablica typu int
int tekst[10]; //10-elementowa tablica typu char (tablica tekstowa)
```

Odwołać się do elementu tablicy można przez indeks do tego elementu np.:

```
int tab2[4]={2,5,6,7};
int a, b;
a=tab2[0];  //zmiennej a przypisz zawartość pierwszego elementu tablicy
            //tab2
b=tab2[3];  //zmiennej b przypisz zawartość ostatniego elementu tablicy
            //tab2
tab2[2]=45; //elementowi tablicy o indeksie 2 (trzeciemu) nadaj wartość 45
```

Uwaga: **Indeksowanie tablicy odbywa się zawsze od 0!** Ostatni element ma indeks równy (rozmiar tablicy) –1!

Przykład:

```
#include <stdio.h>

int main()
{
      int t[4] = {2,5,4,1}; //utworzenie i inicjalizacja tablicy
      int a,i;
      a = t[2];         //do zmiennej a przypisz 3 el. tablicy
      t[0] = 8;         //pierwszemu elementowi tablicy nadaj wart. 8
      for(i=0; i<4; i=i+1)
      {
            printf("%i",t[i]); putchar('\t');
      }
      putchar('\n');
      return 0;
}
```

Tworzona jest 4-elementowa tablica wartości całkowitych typu int. Tablica jest od razu inicjalizowana wartościami 2, 5, 4, 1. Tworzone są dwie zmienne: `a` oraz `i`. Zmiennej `a` nadawana jest wartość trzeciego elementu tablicy. Pierwszy element tablicy uzyskuje wartość 8. Zmiennej `i` nadaje się wartość 0. Dopóki i jest mniejsza od 4:

- wypisywany jest element tablicy o numerze `i`;
- wypisywany jest znak specjalny tabulacji `\t`;
- zwiększana jest wartość zmiennej `i` o 1.

Po zakończeniu wykonywania pętli, na konsolę wprowadzany jest znak przejścia do nowej linii.

1.13 Tablice jednowymiarowe zawierające tekst

Tablice o elementach typu char mogą przechowywać teksty. Tekst w języku C jest ciągiem znaków (typu char) zakończonych liczbą 0. Liczbę 0 w tablicach tekstowych często koduje się przy pomocy specjalnego znaku: `\0`. Jeśli chcemy umieścić w tablicy o elementach typu char pewien tekst, możemy utworzyć tablicę , a następnie przypisać jej stałą tekstową, np.:

```
char tekst1[100] = "To jest tekst";
```

W pamięci w poszczególnych elementach tablicy zapisane zostaną kolejne znaki tekstu, a na koniec znak `\0`: `tekst1[0]=='T'`, `tekst1[1]=='o'`, `tekst1[2]==' '`, ..., `tekst1[12]=='t'`, `tekst1[13]=='\0'`.

Przy inicjalizacji tablicy należy zwrócić uwagę na jej rozmiar. Rozmiar powinien być wystarczający do przechowania tekstu. W przykładowej tablicy tekst1 o rozmiarze 100 elementów, zapamiętany został tekst o długości 13 znaków (w długości tekstu nie uwzględnia się znaku `\0` na końcu każdego tekstu). Podczas przetwarzania tekstów z reguły nie uwzględnia się faktycznej długości tablicy, a raczej długość tekstu w niej zawartego.

Istnieje możliwość wprowadzenia z konsoli tekstu. Może do tego posłużyć funkcja **scanf**, w której jako pierwszy parametr należy podać `"%s"`. Wczytując tekstu funkcją **scanf** nie należy podawać przed nazwą zmiennej znaku `&`:

```
char slowo[30];
scanf("%s", slowo);
```

Tekst wczytany w ten sposób nie uwzględni jednak białych znaków (spacje i tabulacje). Jeśli chcemy wprowadzić całą linię tekstu wraz z białymi znakami, to zamiast funkcji **scanf** można zastosować funkcję **fgets**:

```
char tekst[100];
fgets(tekst,100,stdin);
```

Funkcja **fgets** przyjmuje trzy parametry: zmienną przechowującą tekst, maksymalną liczbę znaków wczytywanego tekstu i nazwę strumienia `stdin`.

Uwaga: Jeśli użytkownik poda tekst składający się zwiększej liczby znaków niż zadeklarowano, to pozostaną one w buforze i kolejne użycie funkcji wczytującej może nie zadziałać jak należy.

Przykład 1:

```
#include <stdio.h>

int main()
{
	char tekst[100];
	printf("Podaj linie tekstu");
	fgets(tekst,100,stdin);
	printf("Wprowadziles tekst:");
	printf("%s",tekst);
	return 0;
}
```

Tworzona jest 100-elementowa tablica do przechowywania znaków typu ASCII. Na ekranie pojawia się napis „Podaj linie tekstu”. Wprowadzony w konsoli tekst jest pobierany do tablicy a następnie wypisywany na konsoli.

Przetwarzanie tekstów zgromadzonych w tablicach tekstowych polega zwykle na analizie ciągu znaków od początku tablicy, aż do znalezienia wartości 0 (odpowiadającej znakowi końca tekstu `\0`).

Przykład 2:

```
#include <stdio.h>

int main()
{
	int i,str_len;
	char tekst[100];
	printf("Podaj linie tekstu: ");
	fgets(tekst,100,stdin);
	str_len=0;
	for(i=0; tekst[i]!=0; i=i+1)
	{
		str_len = str_len +1;
	}
	printf("Dlugosc wprowadzonego tekstu: ");
	printf("%i",str_len);
	return 0;
}
```

Tworzone są dwie zmienne: i oraz `str_len`. Tworzona jest 100-elementowa tablica typu char o nazwie tekst. Użytkownik wprowadza tekst do tablicy jako ciąg znaków ASCII. Wartość zmiennej `str_len` ustawiana jest na 0. Wartość zmiennej `i` ustawiana jest na 0. Dopóki zawartość elementu tablicy wskazywanego przez wartość `i` (`tekst[i]`) jest różna od 0, zawartość zmiennej `str_len` jest zwiększana o 1, a następnie wartość i jest zwiększana o 1. W rezultacie, po zakończeniu wykonywania pętli, w zmiennej `str_len` będzie się znajdować ilość znaków zawartych w tekście – długość tekstu.

1.14 Tablice liczbowe dwuwymiarowe

Tablicę dwuwymiarową można przedstawić jako pewien zestaw tablic 1-wymiarowych np.:

```
float tab1[3][4];
```

Zmienną `tab1` można interpretować jako tablicę 3-elementową złożoną z tablic 4-elementowych. Tablicę trójwymiarową można traktować jako zestaw tablic dwuwymiarowych itd. W przypadku tablicy dwuwymiarowej indeksy można traktować jako numer wiersza i numer kolumny. Dostęp do elementów tablicy uzyskuje się najprościej przez podanie odpowiedniego zestawu indeksów np.:

```
int a[3][2] =
{
	{2,3},
	{4,8},
	{1,4}
};
int c;
```

```
a[0][0]=0; //elementowi o indeksie (0,0) nadaj wartość 0
c=a[0][1]; //zmiennej c przypisz zawartość elementu tablicy a o indeksach
           //(0,1)
```

Przykład:

```
#include <stdio.h>

int main()
{
	float tab[2][3] =
	{
		{2.0, 8.5, -12.0},
		{-23.3, 10.23, 3.44}
	};
	int i,j;
	float max_el;
	max_el = tab[0][0];
	for(i=0; i<2; i++)
	{
		for(j=0; j<3; j++)
		{
			if(tab[i][j] > max_el)
			{
				max_el = tab[i][j];
			}
		}
	}
	printf("Najwiekszy element tablicy: %f", max_el);
	return 0;
}
```

Przykład pokazuje zasadę przetwarzania dwuwymiarowej tablicy element po elemencie. Wyznacza największy element tablicy i wypisuje go na ekranie.

1.15 Definiowanie funkcji

Pojęcie funkcji wprowadzono w języku C w celu umożliwienia tworzenia podprogramów – fragmentów programów, które mają zdefiniowany interfejs z otoczeniem i mogą być wykorzystywane wielokrotnie w obrębie danego programu lub stosowane w wielu pisanych programach. Typowy program w języku C jest zestawem definicji funkcji oraz sposobu ich wywoływania.

Przykład:

```
#include <stdio.h>

int suma(int a, int b)
{
	int c;
	c = a + b;
	return c;
}

int main()
{
	int x,y,s;
	x=6; y=8;
	s=suma(x,y);
	printf("%i",s);
	putchar('\n');
```

```
        s=suma(3,4);
        printf("%i",s);
        putchar('\n');
        return 0;
}
```

Do funkcji przekazuje się dane poprzez jej parametry. Lista parametrów wraz z określeniem ich typów umieszczona jest w nawiasie po nazwie funkcji. Funkcja może przyjmować dowolną liczbę parametrów lub nie przyjmować parametrów w ogóle (wtedy po nazwie funkcji należy wpisać pusty nawias). Przy definiowaniu funkcji określa się również typ danych, jaki funkcja może zwracać (typ danych podawany jest przed nazwą funkcji).

Wartość funkcji zwracamy wpisując ją po słowie `return`. Po wykonaniu instrukcji `return` funkcja natychmiast kończy się i zwraca określoną wartość. Typ `void` oznacza, że funkcja nie zwraca żadnych wartości.

Funkcja w programie rozpoznawana jest przez swoją nazwę. Niezależnie od położenia i liczby zadeklarowanych w programie funkcji, program zawsze rozpoczyna swoje działanie od wywołania funkcji głównej `main`.

Przesłanie danych do funkcji odbywa się przez wypełnienie listy jej parametrów, np.:
`suma(1,4)`

Przejęcie wyniku zwracanego przez funkcję odbywa się przy pomocy operatora przypisania:
`s = suma(23,44)`

Definicje funkcji wywoływanych w funkcji `main` muszą być umieszczone w górnej części kodu programu (powyżej funkcji `main`). Jeśli chcemy umieścić je poniżej (tak, żeby funkcja `main` była na górze), to możemy to zrobić, pod warunkiem wpisania ich deklaracji powyżej.

Przykład:

```
#include <stdio.h>

int suma(int a, int b);


int main()
{
        ...
}
int suma(int a, int b)
{
        ...
}
```

Do funkcji możemy również przekazać tablicę jako parametr. W nagłówku funkcji trzeba wtedy wpisać nazwę tablicy wraz z wymiarem (wymiarami). Przy wywołaniu funkcji podajemy samą nazwę tablicy.

Przykład:
```
void przetworz_tablice(int tab[10])
{
      ...
}

int main()
{
      int tab[10];
      przetworz_tablice(tab);
}
```

Dopuszczalne jest pominięcie w nagłówku funkcji liczby elementów tablicy jednowymiarowej i zamiast tego wpisanie pustych nawiasów kwadratowych:

```
void przetworz_tablice(int tab[])
```

1.16 Struktury

Za pomocą struktury można zdefiniować własny typ danych. Struktura składa się z jednej lub kilku zmiennych składowych (pól), być może różnych typów. Strukturę najpierw definiujemy, a potem tworzymy jej obiekt/obiekty. Możemy to zrobić w następujący sposób:

```
struct Osoba            //definicja struktury
{
      char imie[100];
      int wiek;
};

struct Osoba ania;      //utworzenie obiektu typu struct Osoba
```

Obiektom struktury można przypisać w momencie deklaracji początkowe wartości pól (inicjalizacja obiektu):

```
//utworzenie i inicjalizacja obiektu typu struct Osoba:
struct Osoba ania = {"Ania",35};
```

Poza inicjalizacją do pól struktury można odwołać się przy pomocy operatora „ . ”:
```
ania.wiek = 36;
```

Uwaga:
Nie jest możliwe wykonanie przypisania `ania.imie = "Anna"`. Do modyfikacji pól tekstowych wymagana jest osobna funkcja:
```
strcpy(ania.imie,"Anna");
```
która wprowadza nowy tekst na miejsce starego. Aby z niej skorzystać należy na początku programu dołączyć (dyrektywą `#include`) bibliotekę `string.h` oprócz standardowej `stdio.h`.

2. Proponowane zadania laboratoryjne

1. Co wypisze program?

```
#include <stdio.h>

int main()
{
	int a;
	float b;
	char c;
	a=18;
	b=3.14;
	c='s';
	printf("%i",a);
	putchar('\n');
	printf("%f",b);
	putchar('\n');
	printf("%c",c);
	putchar('\n');
	return 0;
}
```

2. Dany jest program pobierający jedną wartość zmiennoprzecinkową z konsoli, obliczający z niej wartość bezwzględną i wypisujący obliczoną wartość na konsoli:

```
#include <stdio.h>

int main()
{
	float x, y;
	printf("Podaj liczbe zmiennoprzecinkowa:");
	scanf("%f",&x);
	if(x >= 0.0)
	{
		y = x;
	}
	else
	{
		y = -x;
	}
	printf("Wartosc bezwzgledna z wprowadzonej liczby wynosi: %f", y);
	return 0;
}
```

Wzorując się na programie przykładowym:

a) Napisać program, który prosi użytkownika o podanie dwu liczb całkowitych, a następnie wypisuje ich sumę, różnicę, iloraz, iloczyn oraz resztę z dzielenia jednej liczby przez drugą.

b) Napisać program, który prosi użytkownika o podanie jednej liczby całkowitej, a następnie wypisuje na ekranie informację: „Liczba jest parzysta.”, gdy liczba jest parzysta, lub „Liczba jest nieparzysta.”, gdy liczba jest nieparzysta (uwaga: do określania parzystości można posłużyć się operatorem %).

c) Napisać program, który prosi użytkownika o podanie trzech liczb całkowitych, a następnie dokonuje sprawdzenia, czy z tych trzech liczb interpretowanych jako długości odcinków

można zbudować trójkąt (warunek trójkąta mówi, że z trzech odcinków można zbudować trójkąt, jeśli suma każdych dwu odcinków jest większa od długości pozostałego odcinka).

d) Napisać program wczytujący jedną liczbę zmiennoprzecinkową i sprawdzający, czy należy ona do przedziału [2.0, 8.5).

3. Dany jest przykładowy program czytający jeden znak z konsoli i sprawdzający, czy jest to litera a.

```
#include <stdio.h>

int main()
{
	char znak;
	printf("Podaj znak: ");
	znak = getchar();
	if(znak=='a')
	{
		printf("Wpisales litere a.");
	}
	else
	{
		printf("Wpisales litere inna niz a.");
	}
	return 0;
}
```

Wzorując się na programie przykładowym:

a) Napisać program, który odczytuje pojedynczy znak z konsoli, a następnie sprawdza, czy dany znak jest dużą literą i jeśli tak to wypisuje: „Znak jest dużą literą.”, w przeciwnym wypadku program sprawdza, czy znak jest małą literą, i jeśli tak to wypisuje: „Znak jest małą literą.”, w przeciwnym wypadku program wypisuje „Znak nie jest literą.”.
Podpowiedź: znak jest małą literą, jeśli należy do przedziału liczbowego [‘a’,’z’], znak jest dużą literą, jeśli należy do przedziału liczbowego [‘A’,’Z’]. Jeśli liczba x należy do jakiegoś przedziału np. [a,b], to oznacza, że spełnia wyrażenie logiczne: x>=a and x<=b.

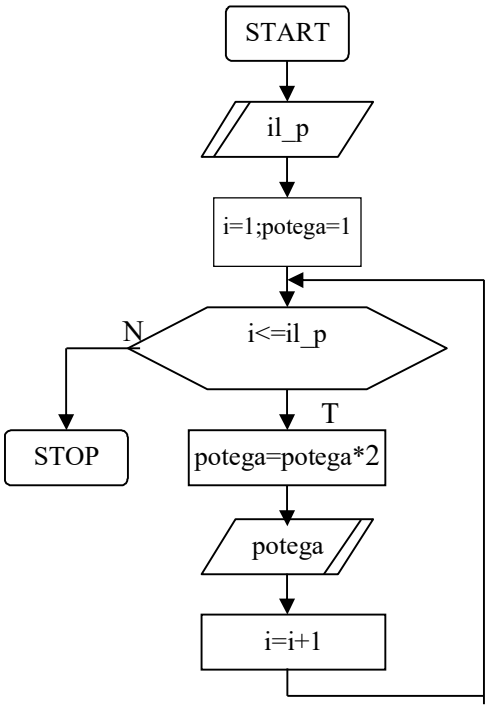
b) Napisać program, który rozpoznaje, czy dany znak jest literą (jakąkolwiek, dużą lub małą), cyfrą, czy innym znakiem.

4. Dany jest przykładowy program, który wypisuje zadaną ilość liczb parzystych:

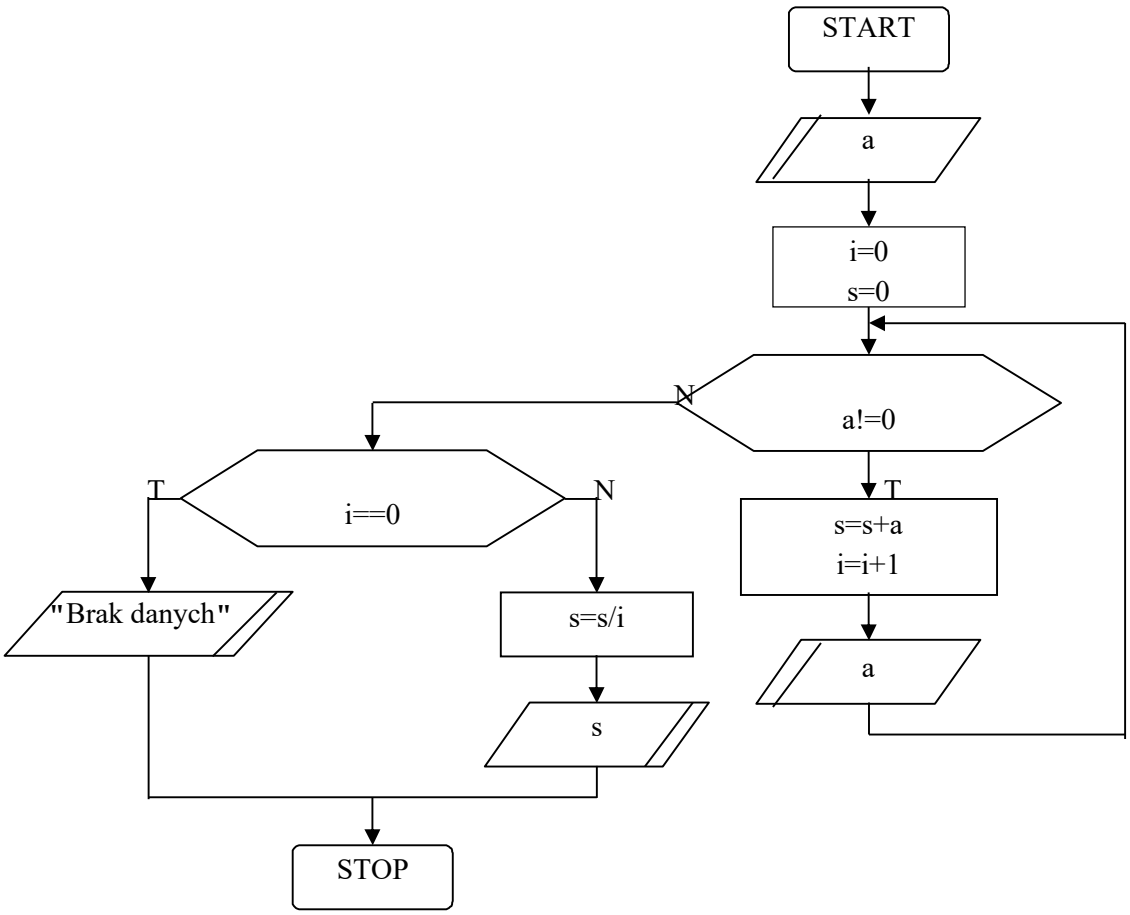
```
#include <stdio.h>

int main()
{
	int i,n,liczba_parzysta;
	printf("Podaj ilosc elementow ciagu do wyswietlenia: ");
	scanf("%i",&n);
	liczba_parzysta = 2;
	i = 0;
	while(i<n)
	{
		printf("%i",liczba_parzysta); putchar('\n');
		liczba_parzysta = liczba_parzysta + 2;
		i = i + 1;
	}
	return 0;
}
```

Wzorując się na przykładowym programie:

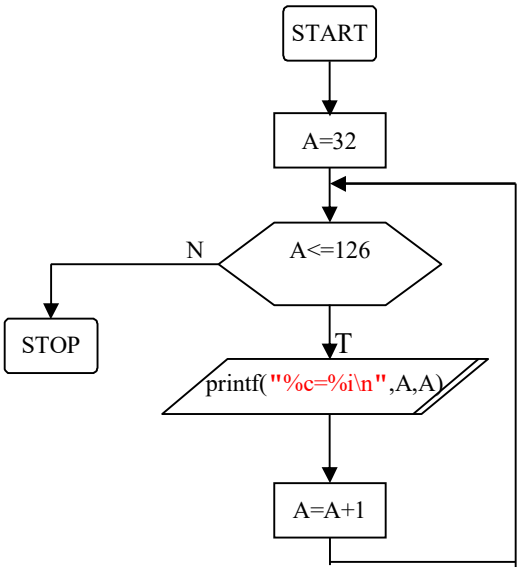
a) Napisać program, który będzie wypisywać zadaną ilość kolejnych potęg liczby 2. Użytkownik będzie podawał ile kolejnych potęg ma zostać wypisane. Zadanie rozwiązać stosując pętlę „while”. Propozycja algorytmu:



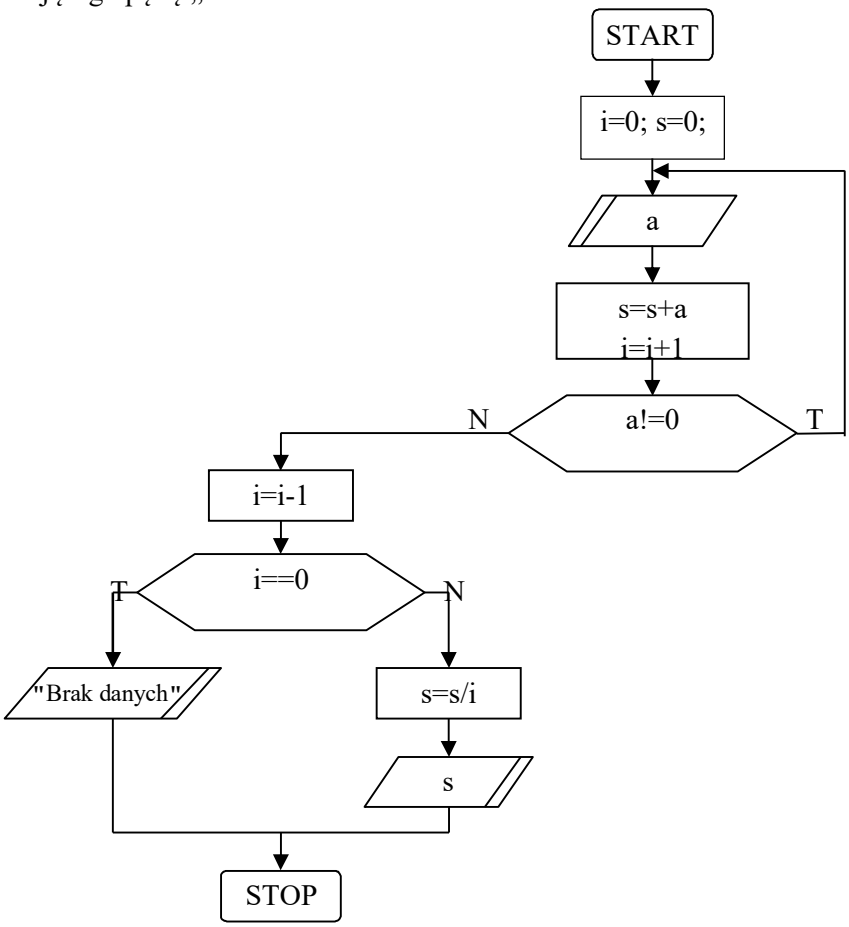
b) Napisać program, który pobiera od użytkownika kolejne liczby zmiennoprzecinkowe i oblicza z nich średnią arytmetyczną. Pobieranie liczb powinno się zakończyć w chwili podania przez użytkownika liczby 0. Zadanie rozwiązać stosując pętlę „while”. Propozycja algorytmu:



c) Napisać program wypisujący na ekranie komputera tablicę znaków ASCII. Podstawowy zestaw znaków ASCII kodujących litery mieści się w przedziale [32,126]. Program powinien wypisać na ekranie pary: znak = kod znaku. Poniżej podano propozycję algorytmu rozwiązującego zagadnienie. Zadanie rozwiązać w dwóch wersjach, stosując pętlę „for” oraz „while”.



d) Poniżej podany jest algorytm wyliczania średniej arytmetycznej z serii danych zakończonych wartością 0 (to samo zagadnienie było rozważane w zadaniu z podpunktu b). Obecny algorytm dostosowano do zastosowania instrukcji „do while”. Należy napisać program według zaproponowanego algorytmu i porównać z programem napisanym według algorytmu stosującego pętlę „while”.



e) Zaproponować rozwiązanie zadania z wypisywaniem kolejnych potęg liczby 2 przy pomocy pętli „for” (treść zadania i algorytm zaproponowano w podpunkcie a)).

5. Dany jest program wczytujący z konsoli pięć liczb do tablicy `tab` i obliczający, ile z elementów tablicy jest parzystych.

```
#include <stdio.h>

int main()
{
	int tab[5];
	int i;
	int ile_parzystych=0;

	printf("Wypelnij tablice: ");
	for(i=0;i<5;i++)
	{
		printf("t %i=",i);
		scanf("%i",&tab[i]);

		if(tab[i]%2 == 0)
		{
			ile_parzystych++;
		}
	}
	printf("W tablicy jest %i liczb parzystych.", ile_parzystych);
	return 0;
}
```

a) należy uruchomić i przetestować program na różnych seriach danych;

b) na podstawie programu, opracować własny, obliczający ile elementów tablicy mieści się w zamkniętym przedziale [0, 22].

6. Biorąc na wzór programy omówione w punkcie 1.13 (str. 10-11), opracować własny:

a) wyliczający:

- liczbę wystąpień dużych liter w tekście;
- liczbę wystąpień małych liter w tekście;
- liczbę wystąpień liter w tekście;
- liczbę wystąpień cyfr w tekście;
- liczbę białych znaków (spacja, tabulacja) w tekście.

b) wyliczający liczbę wystąpień zadanej litery w tekście.

7. Biorąc na wzór program omówiony w punkcie 1.14 (str. 12), opracować własny, wyszukujący w dwuwymiarowej tablicy liczbę nieujemnych i ujemnych wartości.

8. Dany jest przykładowy program:

```
#include <stdio.h>

//tu wstaw definicję funkcji f_abs
```

```
int main()
{
	float a, w_bz;
	printf("Podaj liczbe zmiennoprzecinkowa: ");
	scanf("%f",&a);
	w_bz=f_abs(a);
	printf("|a|=");
	printf("%f",w_bz);
	return 0;
}
```

Napisać funkcję `f_abs`, która zwraca wartość typu float i przyjmuje jeden parametr typu float. Funkcja powinna wyliczyć wartość bezwzględną z wprowadzanej do niej liczby zmiennoprzecinkowej i zwrócić ją. Przykładowy program, który podano powyżej prosi użytkownika o podanie liczby i wywołuje funkcję `f_abs` do obliczenia wartości bezwzględnej.

9. Dany jest program:

```
#include <stdio.h>

//tu wstaw definicję funkcji czy_duza

int main()
{
	char tekst[100];
	int i, l_duza, tmp;
	printf("Podaj linie tekstu: ");
	fgets(tekst,100,stdin);
	l_duza=0;
	for(i=0; tekst[i]!=0; i=i+1)
	{
		tmp = czy_duza(tekst[i]);
		if(tmp == 1)
		{
			l_duza = l_duza + 1;
		}
	}
	printf("W podanym tekscie bylo: ");
	printf("%i",l_duza);
	printf(" duzych liter.");
	return 0;
}
```

Napisać funkcję `czy_duza`, która zwraca wartość typu int i przyjmuje jeden parametr typu char. Funkcja ma zwracać wartość 1, gdy jej parametr jest dużą literą lub 0 w przeciwnym wypadku. Przykładowy program, który podano powyżej stosuje funkcję `czy_duza` do wyliczenia liczby dużych liter we wprowadzonej linii tekstu.

10. Dany jest program:

```
#include <stdio.h>

//tu wstaw definicję funkcji na_duza

int main()
{
	char tekst[100];
	int i;
	printf("Podaj linie tekstu: ");
	fgets(tekst,100,stdin);
```

```
        for(i=0; tekst[i]!=0; i=i+1)
        {
              tekst[i] = na_duza(tekst[i]);
        }
        printf("Tekst po przetworzeniu: ");
        printf("%s",tekst);
        return 0;
}
```

Napisać funkcję `na_duza`, która zwraca wartość typu char i przyjmuje jeden parametr typu char. Funkcja powinna sprawdzać, czy znak jest małą literą i jeśli tak, to zamienić go na dużą literę. Pozostałe znaki pozostawiamy bez zmian. Przykładowy program, który podano powyżej stosuje funkcję `na_duza` do przekształcenia linii tekstu w taki sposób, aby wszystkie małe litery tekstu zapisane były jako duże.

11. Dany jest program:

```
#include <stdio.h>

//tu wstaw definicję funkcji trojkat_prostokatny

int main()
{
    float a, b, c;
    printf("Podaj dlugosc boku pierwszego: ");
    scanf("%f", &a);
    printf("Podaj dlugosc boku drugiego: ");
    scanf("%f", &b);
    printf("Podaj dlugosc boku trzeciego: ");
    scanf("%f", &c);
    if(a <= 0 || b <= 0 || c <= 0)
    {
        printf("Podano niedodatnia dlugosc przynajmniej jednego boku.");
    }
    else
    {
        trojkat_prostokatny(a,b,c);
    }
    return 0;
}
```

Napisać funkcję `trojkat_prostokatny`, która nie zwraca żadnej wartości i przyjmuje trzy parametry typu float oznaczające długości boków trójkąta. Funkcja powinna sprawdzić, czy z podanych długości da się utworzyć trójkąt prostokątny i wypisać „Jest to trójkąt prostokątny” lub „Nie jest to trójkąt prostokątny”. Funkcja zakłada, że zostały podane liczby dodatnie, więc nie trzeba tego sprawdzać.

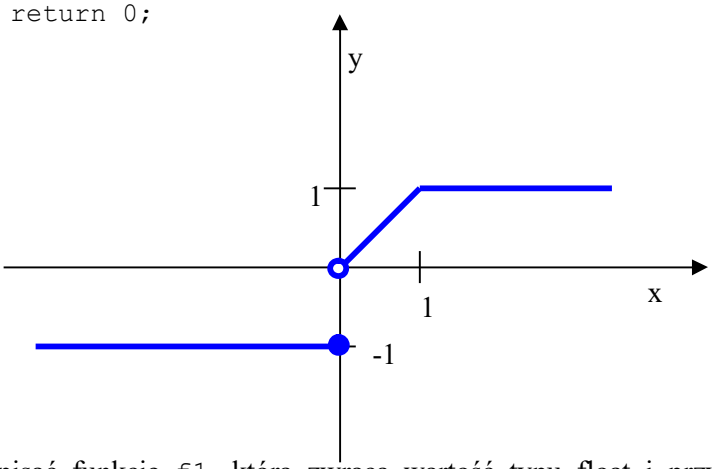
Należy pamiętać, że boki mogą zostać podane w dowolnej kolejności (nie wiemy, która wartość jest przeciwprostokątną), co oznacza, że musimy sprawdzić trzy warunki dotyczące twierdzenia Pitagorasa z różnymi kolejnościami boków.

Przykładowy program, który podano powyżej, prosi użytkownika o wczytanie trzech długości boków i sprawdza, czy są one dodatnie. Jeśli tak, to wywołuje funkcję `trojkat_prostokatny`. Przykładowe długości boków trójkąta prostokątnego do sprawdzenia programu:

- 3, 4, 5
- 20, 21, 29

12. Dany jest program i wykres pewnej funkcji:

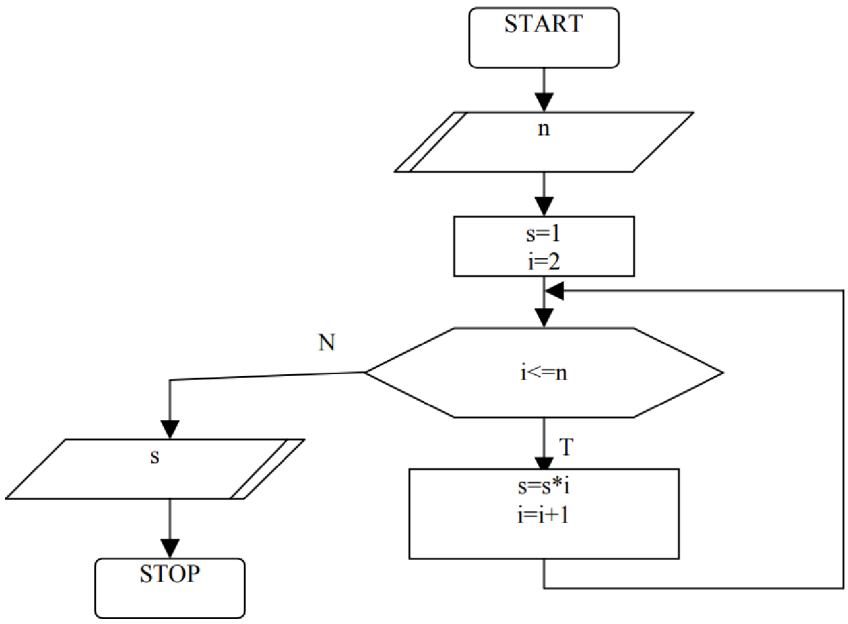
```
#include <stdio.h>

//tu wstaw definicję funkcji f1

int main()
{
	float a=-3.0, b=0.45, c=11.0, f1a, f1b, f1c;
	f1a=f1(a);
	f1b=f1(b);
	f1c=f1(c);
	printf("\n f1(%f) = %f",a, f1a);
	printf("\n f1(%f) = %f",b, f1b);
	printf("\n f1(%f) = %f",c, f1c);
	return 0;
}
```



Napisać funkcję `f1`, która zwraca wartość typu float i przyjmuje jeden parametr typu float. Funkcja powinna zwracać wartości zgodne z danym wykresem.

13. Napisz funkcję `silnia`, która zwraca wartość typu int i przyjmuje jeden parametr typu int o nazwie `n`. Funkcja powinna obliczyć i zwrócić silnię z podanego parametru (zakładamy, że użytkownik podał parametr będący liczbą dodatnią).

    Następnie w funkcji głównej wywołaj funkcję `silnia` z parametrem `n` podanym przez użytkownika oraz wypisz wynik. Poniżej zamieszczono algorytm obliczania silni:



14. Dany jest program:

```
#include <stdio.h>

//tu wstaw definicję funkcji sumuj_parzyste

int main()
{
    int tab[2][4];
    int suma;
    printf("Wypelnij tablice 2D o wymiarach 2x4.\n");
    for(int i=0; i<2; i++)
    {
        for(int j=0; j<4; j++)
        {
            printf("Podaj liczbe [%i][%i]: ", i, j);
            scanf("%i", &tab[i][j]);
        }
    }
    suma = sumuj_parzyste(tab);
    printf("Suma parzystych elementow = %i", suma);
    return 0;
}
```

Napisać funkcję `sumuj_parzyste`, która zwraca wartość typu int i przyjmuje jeden parametr będący tablicą dwuwymiarową typu int o wymiarach 2x4 (2 wiersze, 4 kolumny). Funkcja powinna obliczyć sumę wszystkich parzystych elementów tablicy (elementy nieparzyste pomijamy).

Przykładowy program, który podano powyżej, prosi użytkownika o wczytanie wszystkich elementów tablicy, a następnie wywołuje funkcję `sumuj_parzyste` i wypisuje wynik.

15. Dany jest przykładowy program, który w oparciu o strukturę w języku C tworzy prostą 2–elementową bazę danych osób:

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

struct Osoba  // definicja struktury
{
      char imie[100];
      int wiek;
};

int main()
{
      struct Osoba ania = {"Ania",35};
      struct Osoba beata;
      struct Osoba nieznana;

      // wprowadzanie i wczytywanie danych
      strcpy(beata.imie,"Beata");
      beata.wiek = 32;

      printf("Podaj imie: ");
      scanf("%s",nieznana.imie); //uwaga: wczytując tekst scanfem piszemy
      printf("Podaj wiek: ");    //%s i nie piszemy & przed nazwą zmiennej
      scanf("%i",&nieznana.wiek);
```

```
    // wypisywanie danych
    printf("Pola bazy:\n");
    printf("%s",ania.imie);//wypisując tekst printfem również piszemy %s
    printf(" %i",ania.wiek);
    putchar('\n');

    printf("%s",beata.imie);
    printf(" %i",beata.wiek);
    putchar('\n');

    printf("%s",nieznana.imie);
    printf(" %i",nieznana.wiek);

    return 0;
}
```

Podany program rozbudować w taki sposób, aby struktura `Osoba` zawierała dodatkowo wzrost (liczba zmiennoprzecinkowa). Następnie należy wpisać przykładowy wzrost Ani, Beaty i Nieznanej analogicznie do pozostałych danych.

16. Zdefiniuj strukturę `Trapez` zawierającą trzy zmienne typu zmiennoprzecinkowego: `podstawa1`, `podstawa2` i `wysokosc`. Następnie napisz funkcję `porownanie_trapezow`, która nie zwraca żadnej wartości i przyjmuje dwa parametry typu `Trapez`. Funkcja powinna obliczyć pola obu trapezów i wypisać informację „Pierwszy trapez jest większy.” lub „Drugi trapez jest większy.” w zależności od tego, który z trapezów ma większe pole.

    W funkcji głównej utwórz dwa obiekty trapezów z danymi wczytanymi przez użytkownika z konsoli, a następnie porównaj je za pomocą napisanej funkcji.

17. Zmodyfikuj poprzednie zadanie tak, aby funkcja `porownanie_trapezow` przyjmowała jako parametr tablicę 5-elementową typu `Trapez`. Funkcja powinna zwrócić numer indeksu trapezu, którego pole jest największe.

    W funkcji głównej przetestuj działanie napisanej funkcji.