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

Opracowali: Sławomir Samolej, Andrzej Bożek
Politechnika Rzeszowska,
Katedra Informatyki i Automatyki,
Rzeszów, 2009.

1.1. 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.

1.1.1 Podstawowe typy danych

W języku C zdefiniowano 4 podstawowe typy danych:

char mała liczba całkowita, zwykle służąca do przechowywania zakodowanych w standardzie ASCII znaków;
int liczba całkowita podstawowej długości;
float liczba zmiennopozycyjna podstawowej długości;
double liczba o rozszerzonej długości.

1.1.2 Tworzenie zmiennych w języku C

Instrukcja utworzenia zmiennej w języku C zawiera najpierw określenie typu zmiennej, następnie jej nazwy.
Przykład:

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

W przykładowym fragmencie programu utworzono 2 zmienne **a** i **b**. Zmienna **a** jest typu **int**, a zmienna **b** typu **float**.
Uwaga 1:
Nazwa zmiennej nie może się zaczynać od cyfry. Nie można tworzyć nazw zmiennych z polskim znakami diakrytycznymi. Nazwa ma być 1 wyrazem (bez spacji).
Uwaga 2:
Zmienne w języku C można tworzyć „na zewnątrz” wszystkich funkcji programu albo na początku bloku instrukcji (zaraz po nawiasie klamrowym: „{”).

1.1.3 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**.

1.1.4 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. w [1],[4].
Uwaga:
Aby można było skorzystać ze standardowych mechanizmów wejścia-wyjścia w języku C, należy na początku treści programu wskazać plik z odpowiednimi nagłówkami funkcji wej/wyj:

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

Do wyprowadzania danych na konsolę służą funkcje **printf**(), **putchar**(), **puts**().

Funkcję **printf**() w podstawowej konfiguracji można zastosować do wypisywania pojedynczych wartości liczbowych.
Przykład:

```
{
	int a;
	a=5;
	printf("%d",a);
}
```

Utworzonej zmiennej całkowitej **a** nadano wartość **5**, a następnie wypisano wartość tej zmiennej na ekranie w postaci dziesiętnej. 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:
%d – zmienna całkowita w postaci dziesiętnej;
%x – zmienna całkowita w postaci szesnastkowej;
%f - zmienna zmiennopozycyjna w postaci dziesiętnej.

Funkcja **putchar**() w podstawowym zastosowaniu służy do wyprowadzania wartości 1 zmiennej w postaci zdekodowanej na znak ASCII.
Przykład:

```
{
	char c;
```

```
        c='a';
        putchar(c);
        putchar('\n');
}
```

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 '\n', czyli wymuszono przejście do następnego wiersza.

Funkcja **puts**() w podstawowym zastosowaniu służy do wyprowadzania tekstu na ekranie.
Przykład:

```
{
        puts("To jest tekst");
}
```

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

Do podstawowego pobierania danych z konsoli służą funkcje **scanf**() oraz **getchar**().

Funkcja **scanf**() w podstawowym trybie swojej pracy może służyć do pobierania 1 wartości liczbowej z konsoli.
Przykład:

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

Do utworzonej zmiennej zmiennopozycyjnej 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 1 znaku z konsoli.
Przykład:

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

Do zmiennej znak zostanie wprowadzony kod litery podanej na konsoli.

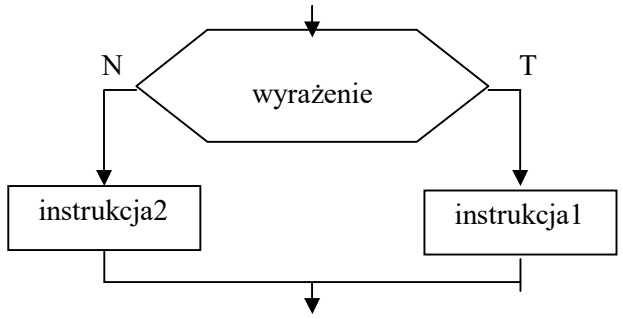
1.1.5 Operatory porównania

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?

1.1.6 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,b;
      a=4;
      b=5;
      if (a > b)                //jeśli a > b
            { puts("A>B");      // to pisz: A>B
              b = 3;            // b nadaj wartość 3
            }
      else
            {puts("A<=B");}     // w przeciwnym wypadku pisz A<=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”.

1.1.7 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; printf("%d",a); putchar('\n'); //zwiększenie a o 1
c=b%6; printf("%d",c); putchar('\n'); //reszta z dzielenia b przez 6
}
```

1.1.8 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”

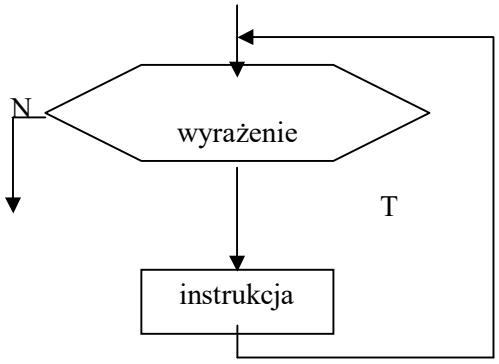
Przykład:

```
{       int a;
        scanf("%d",&a);
        if (a > 6 && a < 10)          //jeśli a > 6 i a < 10
                puts("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.

1.1.9 Instrukcja 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:

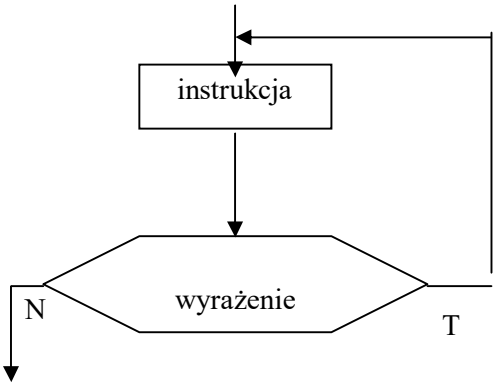
```
{
	char a;
	a = 10;
	while(a>0)
	{
		a=a-1;
		printf("%d",a);
		putchar('\n');
	}
	a = 8;
}
```

Tworzona jest zmienna całkowita **a**. Nadawana jest jej wartość 10. Dopóki wartość przechowywana w zmiennej **a** jest większa od 0:
- wartość a jest zmniejszana o 1;
- wartość a jest wypisywana na konsoli.

Po zakończeniu wykonywania pętli zmiennej **a** nadaje się wartość 8.

1.1.10 Instrukcja 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.

Przykład:

```
{
	int a,b;
	a = 10;
	do
	{
		a=a-1;
		if(a%2 == 1)
		{
			printf("%d",a);
			putchar('\n');
		}
		else
		{
			b = 2 * a;
			printf("%d",b);
			putchar('\n');
		}
	}
	while(a>0);
}
```

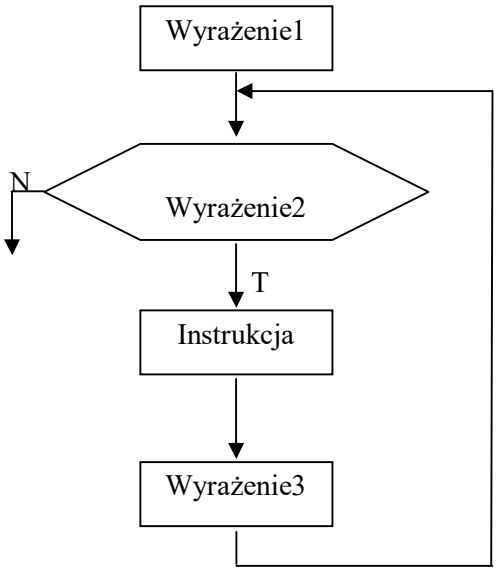
Tworzone są 2 zmienne: a i b. Zmiennej a nadaje się wartość 10. Rozpoczyna się cykliczne wykonywanie instrukcji:

- zmniejsz a o 1;
- jeśli reszta z dzielenia a przez 2 wynosi 1, to:
  - wypisz a;
- w przeciwnym wypadku:
  - b nadaj wartość 2 * a;
  - wypisz b.

Powyższe instrukcje będą wykonywane dopóki wartość zmiennej a będzie większa od 0.

1.1.11 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:

```
{
	char a,b;
	for(a=10;a>0;a=a-1)
	{
		if(a%2==1)
		{	printf("%d",a);
			putchar('\n');
		}
		else
		{	b=2*a;
			printf("%d",b);
			putchar('\n');
		}
	}
}
```

1.1.12 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 elementów typu int
char tekst[10]; // 10-elementowa tablica elementów 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>

void main(void)
{
	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("%d",t[i]); putchar('\t');
	}
	putchar('\n');
}
```

Tworzona jest 4-elementowa tablica wartości całkowitych typu int. Tablica jest od razu inicjalizowana wartościami 2, 5, 4, 1. Tworzone są 2 zmienne: a oraz i. Zmiennej a nadawana jest wartość 3 el. 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 tabulacji;
- zwiększana jest wartość zmiennej i o 1.

Po zakończeniu wykonywania pętli, na konsolę wprowadzany jest znak nowego wiersza.

1.1.13 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 '\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 linii tekstu. Może do tego posłużyć funkcja **gets**();
Przykład 1:

```
#include <stdio.h>

void main(void)
{
	char tekst[100];
	puts("Podaj linie tekstu");
	gets(tekst);
	puts("Wprowadziles tekst:");
	puts(tekst);
}
```

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 (znaku końca tekstu).
Przykład 2:

```
#include <stdio.h>

void main(void)
{
	int i,str_len;
	char tekst[100];
	puts("Podaj linie tekstu");
	gets(tekst);
	str_len=0;
	for(i=0;tekst[i]!=0;i=i+1)
	{
		str_len = str_len +1;
	}
	puts("Dlugosc wprowadzonego tekstu:");
	printf("%d",str_len);
	putchar('\n');
}
```

Tworzone są 2 zmienne i oraz str_len. Tworzona jest 100-elementowa tablica „tekst" wartości typu char. Użytkownik wprowadza do tablicy tekst 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.

1.1.14 Tablice liczbowe 2-wymiarowe

Tablicę 2-wymiarową można przedstawić jako pewien zestaw tablic 1-wymiarowych np.:

float tab1[3][4];

tab1 można interpretować jako tablicę 3-elementową złożoną z tablic 4-elementowych.
Tablicę 3-wymiarową można traktować jako zestaw tablic 2-wymiarowych itd.
W przypadku tablicy 2-wymiarowej 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>

void main(void)
{
      float tab[2][3]={ { 2.0f,  8.5f, 12.0f},
                        {-23.3f, 10.23f,  3.44f}
                    };
      int i,j;
      float max_el;
      max_el = tab[0][0];
      for(i=0;i<2;i=i+1)
      {     for(j=0;j<3;j=j+1)
            {
                  if(max_el<tab[i][j])
                  {
                        max_el = tab[i][j];
                  }

            }
      }
      puts("Najwiekszy el. tablicy:");
      printf("%f",max_el);
      putchar('\n');
}
```

Przykład pokazuje zasadę przetwarzania 2-wymiarowej tablicy element po elemencie. Wyznacza największy element tablicy i wypisuje go na ekranie.

1.1.15 Struktury

Struktura jest obiektem złożonym z jednej lub kilku zmiennych, być może różnych typów.
Struktury można deklarować w następujący sposób:

```
struct wiersz_bazy            // deklaracja struktury
{
      char imie[100];
      int wiek;
};

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

Elementom struktury można przypisać początkowe wartości w momencie inicjalizacji:

```
struct wiersz_bazy            // deklaracja struktury
{
      char imie[100];
      int wiek;
};
// utworzenie i inicjalizacja obiektu typu struct wiersz_bazy:
struct wiersz_bazy ania={”Ania”,35};
```

Poza inicjalizacją do pól struktury można odwołać się przy pomocy operatora „ . ”:

```
struct wiersz_bazy            // deklaracja struktury
{
      char imie[100];
      int wiek;
};

struct wiersz_bazy ania={”Ania”,35};
ania.wiek=23;
```

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.

1.1.16 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);       //deklaracja funkcji
```

```
void main(void)
{
	int x,y,s;
	x=6;
	y=8;
	s=suma(x,y);
	printf("%d",s);
	putchar('\n');
	s=suma(3,4);
	printf("%d",s);
	putchar('\n');
}

int suma(int a, int b)  //definicja funkcji
{
	int c;
	c=a+b;
	return c;
}
```

Do funkcji przekazuje się dane poprzez jej parametry (lista parametrów wraz z określeniem ich typów umieszczona jest w nawiasie po nazwie funkcji). Przy definiowaniu funkcji określa się również typ danych, jaki funkcja może zwracać (typ danych podawany jest przed nazwą funkcji). Funkcja w programie rozpoznawana jest przez swoją nazwę. Niezależnie od położenia i ilości zadeklarowanych w programie funkcji, program zawsze rozpoczyna swoje działanie od wywołania funkcji 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)

1.2. Proponowane zadania laboratoryjne

1. Co wypisze program?

```
#include <stdio.h>

void main(void)
{
	int a;
	float b;
	char c;
	a=18;
	b=3.14f;
	c='s';
	printf("%d",a); putchar('\t');
	printf("%x",a); putchar('\t');
	putchar('\n');
	printf("%f",b);
	putchar('\n');
	putchar(c);
	putchar('\n');
}
```

2. Dany jest program pobierający 1 wartość zmiennopozycyjną z konsoli, obliczający z niej wartość bezwzględną i zwracający obliczoną wartość na konsolę:

```
#include <stdio.h>

void main(void)
{
	float x;
	float y;
	puts("Podaj liczbe zmiennopozycyjna:");
	scanf("%f",&x);
	if(x >= 0.0)
	{
		y = x;
	}
	else
	{
		y = -x;
	}
	puts("Wartosc bezwzgledna z wprowadzonej liczby wynosi:");
	printf("%f",y);
	putchar('\n');
}
```

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

d) Napisać program wczytujący 1 liczbę zmiennoprzecinkową i sprawdzający, czy należy ona do przedziału [2.0, 8.5).

3. Dany jest przykładowy program czytający 1 znak z konsoli i sprawdzający, czy jest to litera a.

```
#include <stdio.h>

void main(void)
{
	char znak;
	znak=getchar();
	if (znak=='a')
	{
		puts("Wpisales litere a");
	}
	else
	{
		puts("Wpisales litere inna niz a");
	}
}
```

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 znak jest małą literą wypisuje: „znak ... jest małą literą”, w przeciwnym wypadku program wypisuje „znak nie jest literą” (Uwaga: 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 i x<=b).

b) Napisać program, który rozpoznaje, że dany znak jest literą lub cyfrą lub innym znakiem.

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

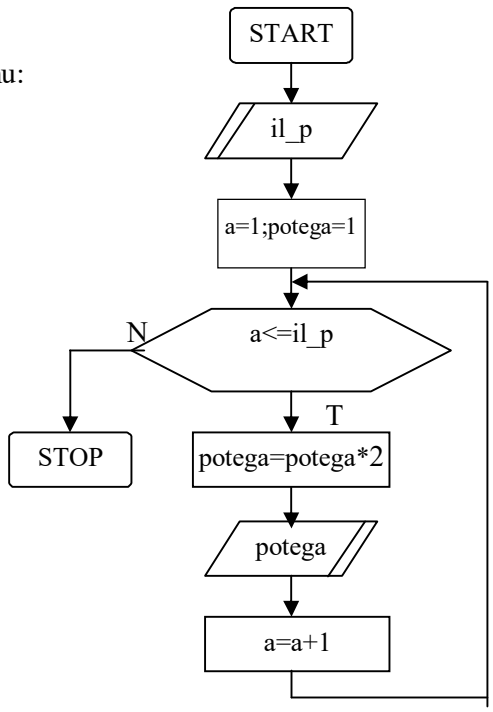
```
#include <stdio.h>

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

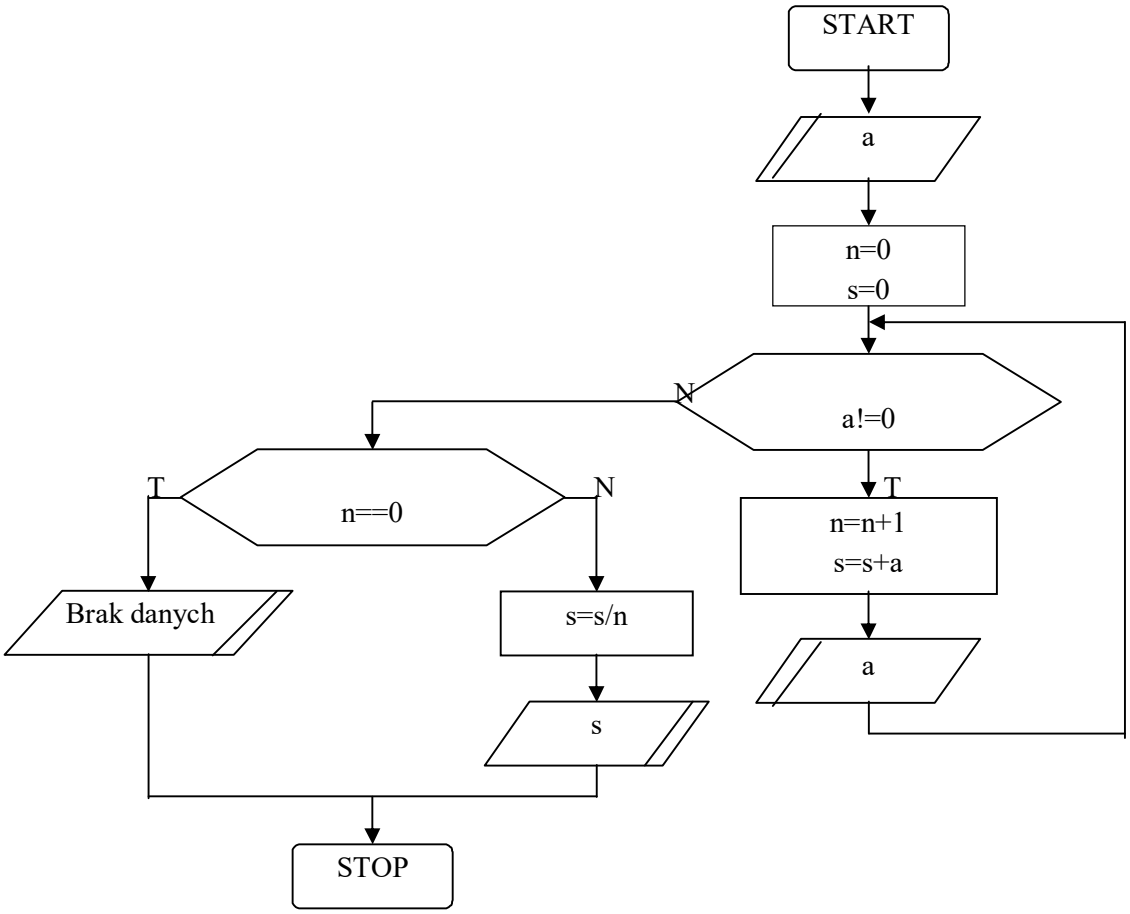
Wzorując się na przykładowym programie:

a) Napisać program, który wypisywać będzie zadaną ilość kolejnych potęg liczby 2. Użytkownik będzie podawał ile kolejnych potęg ma zostać wypisane.

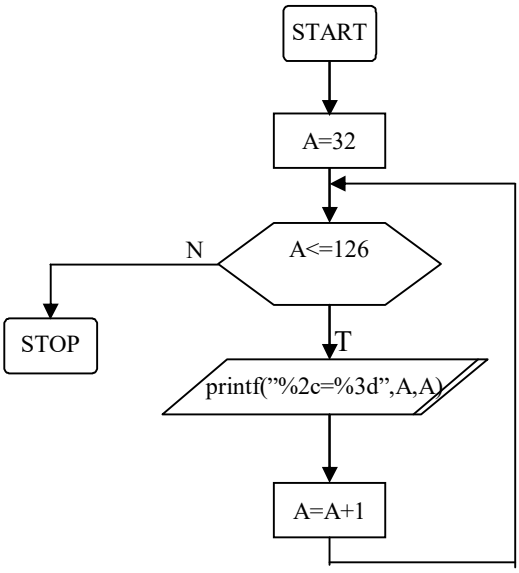
Propozycja algorytmu:



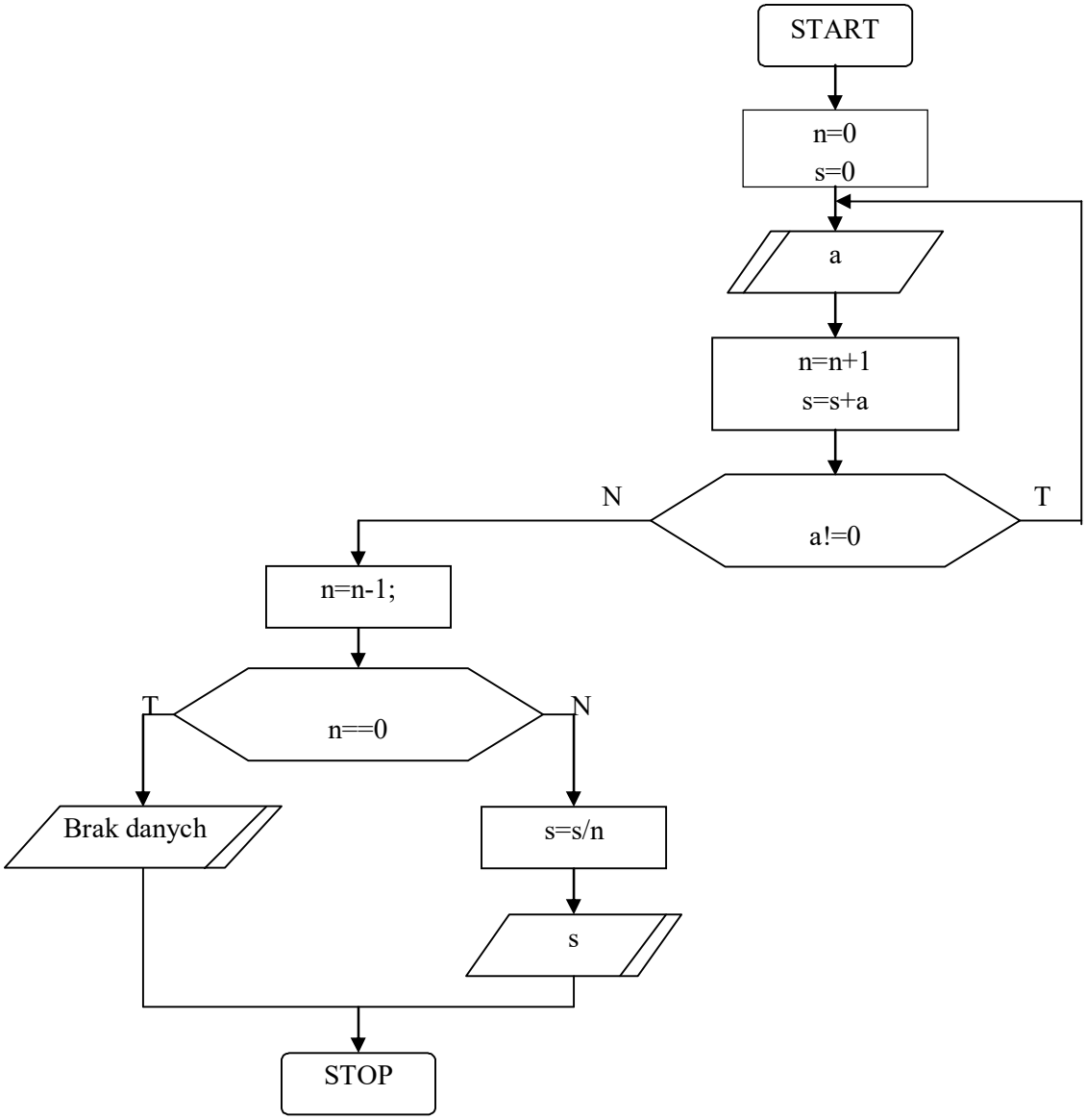
b) Napisać program, który pobiera od użytkownika kolejne liczby zmiennopozycyjne i oblicza z nich średnią arytmetyczną. Pobieranie liczb powinno się zakończyć w chwili podania przez użytkownika liczby 0.
Propozycja algorytmu:



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 stosując pętle „while" i „for".



d) Poniżej podany jest algorytm wyliczania średniej arytmetycznej z serii danych zakończonych wartością 0.0 (to samo zagadnienie było rozważane w poprzednim zadaniu). 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 5 liczb do tablicy tab i obliczający, ile z elementów tablicy jest parzystych.

```
#include <stdio.h>

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

	puts("Wypelnij 5 elementowa tablice liczbami calowitymi:");
	for(i=0;i<5;i=i+1)
	{
		putchar('t'); printf("%d",i); putchar('=');
		scanf("%d",&tab[i]);
	}
	ile_parzystych=0;
	for(i=0;i<5;i=i+1)
	{
		int tymcz;
		tymcz=tab[i]%2;
		if(tymcz == 0)
		{
			ile_parzystych = ile_parzystych + 1;
		}
	}
	puts("We wprowadzonej tablicy bylo");
	printf("%d",ile_parzystych);
	puts(" liczb parzystych.");
}
```

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

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

6. Biorąc na wzór program omówiony w punkcie 1.1.13 instrukcji, opracować własny:

   a) wyliczający ilość wystąpień dużych liter w tekście;

   b) wyliczający ilość wystąpień małych liter w tekście;

   c) wyliczający ilość wystąpień liter w tekście;

   d) wyliczający ilość wystąpień cyfr w tekście;

   e) wyliczający ilość białych znaków w tekście;

   f) wyliczający ilość wystąpień zadanej litery w tekście.

7. Biorąc na wzór program omówiony w punkcie 1.1.14 instrukcji, opracować własny, wyszukujący w dwuwymiarowej tablicy ilość nieujemnych i ujemnych wartości.

8. Dany jest przykładowy program, który w oparciu o strukturę w języku C tworzy prostą 3 – elementową bazę danych Pań:

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

struct wiersz_bazy            // deklaracja struktury
{
```

```
	char imie[100];
	int wiek;
};

void main(void)
{
	struct wiersz_bazy Ania={"Ania",35};
	struct wiersz_bazy Beata;
	struct wiersz_bazy Nieznana;
	int tmp;

	strcpy(Beata.imie,"Beata");
	Beata.wiek=32;

	puts("Podaj imie:");
	gets(Nieznana.imie);
	puts("Podaj wiek:");
	scanf("%d",&tmp);
	Nieznana.wiek=tmp;

	puts("Pola bazy:");
	puts(Ania.imie);
	printf("%d",Ania.wiek);
	putchar('\n');

	puts(Beata.imie);
	printf("%d",Beata.wiek);
	putchar('\n');

	puts(Nieznana.imie);
	printf("%d",Nieznana.wiek);
	putchar('\n');
}
```

Podany program rozbudować w taki sposób, aby wiersz bazy danych zawierał dodatkowo nazwisko oraz wzrost.

9. Dany jest przykładowy program:

```
#include <stdio.h>

float fabs(float x);    //deklaracja funkcji

void main(void)
{
	float a, w_bz;
	puts("Podaj liczbę zmiennopozycyjna:");
	scanf("%f",&a);
	w_bz=-1.0;
	w_bz=fabs(a);
	puts("|a|=");
	printf("%f",w_bz);
}

float fabs(float x)     //definicja funkcji
{
}
```

Uzupełnić treść funkcji fabs, w taki sposób, aby wyliczała ona wartość bezwzględną z wprowadzanej do niej liczny zmiennopozycyjnej i zwracała ją.

10. Dany jest program:

```
#include <stdio.h>

int czy_duza(char t);

void main(void)
{
	char tekst[100];
	int i, l_duza, tmp;
	puts("Podaj 1 linie tekstu:");
	gets(tekst);
	l_duza=0;
	tmp=-1;
	for(i = 0; tekst[i] != 0; i = i + 1)
	{
		tmp = czy_duza(tekst[i]);
		if(tmp == 1) l_duza = l_duza + 1;
	}
	puts("W podanym tekscie bylo:");
	printf("%d",l_duza);
	puts("duzych liter");
}

int czy_duza(char t)
{

}
```

Zaproponować treść funkcji “czy_duza”, która ma zwracać wartość 1, gdy jej parametr wywołania jest dużą literą lub 0 w przeciwnym wypadku. Przykładowy program, który podano powyżej stosuje funkcję „czy_duza” do wyliczenia ilości dużych liter we wprowadzonej linii tekstu.

11. Dany jest program:

```
#include <stdio.h>

char na_duza(char t);

void main(void)
{
	char tekst[100];
	int i, l_duza, tmp;
	puts("Podaj 1 linie tekstu:");
	gets(tekst);
	for(i = 0; tekst[i] != 0; i = i + 1)
	{
		tekst[i] = na_duza(tekst[i]);
	}
	puts("Tekst po przetworzeniu:");
	puts(tekst);
}

char na_duza(char t)
{

}
```

Zaproponować treść funkcji “na_duza”, która ma zamieniać wszystkie małe litery na duże, pozostałe zaś znaki pozostawiać 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.

12. Dany jest program i wykres pewnej funkcji:

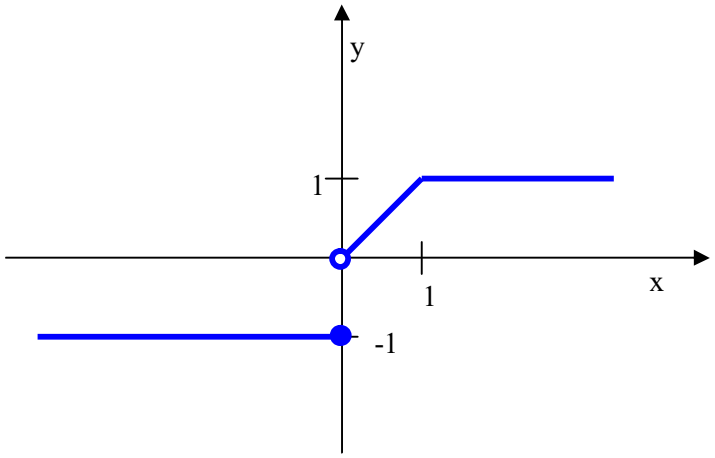
```
#include <stdio.h>

float f1(float x);      //deklaracja funkcji

void main(void)
{
      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);
}

float f1(float x)
{

}
```



Należy uzupełnić funkcję f1 w taki sposób, aby obliczała i zwracała wartości zgodne z danym wykresem funkcji.

1.3. Literatura

[1] W. Kernighan, D.M. Ritichie, „Język ANSI C”, WNT Warszawa 1994.
[2] R. K. Konieczny, „Wprowadzenie do programowania w języku C”, Intersofland Warszawa 1993.
[3] C. Delannoy, „Ćwiczenia z Języka C”, WNT 1993.
[4] H. Schildt , „Programowanie C”, Wydawnictwo RM, 2002.