Example 8

Problem:

Write the program printing ASCII codes for letters and digits.

Discussion:

Given: Latin letters and digits
Input data: no input
Output data: Latin letters and digits and their ASCII codes

Algorithm:

Print letters and their ASCII codes in the first loop and digits and their ASCII codes in the second loop.


```
/*File ASCII_codes;
    Program writes ASCII codes for letters and digits*/

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

int main()
{
    char ch;
    printf("Program writes ASCII codes for letters and digits\n");
    for(ch='a'; ch<='z'; ch++)
          printf("ASCII code for the letter %c is %d\n", ch, ch);
    for(ch='0'; ch<='9'; ch++)
          printf("'ASCII code for the digit %c is %d\n", ch, ch);

   getch();        // conio.h
   return 0;
}
```

Data conversion

Functions in stdlib.h library:

atoi(char_array) – converts text into integer number; if the string contains a decimal place, the number will be truncated.

atof(char_array) - converts text into floating point number of double type.

atol(char_array) – converts text into integer number of **long int** type.

If the string starts with an invalid number, the functions return zero.

Examples

```
# include <stdio.h>
# include <stdlib.h>
   …
char  text [20];
int     intvar;
double  floatvar;

gets(text);
intvart = atoi(text);
floatvart = atof(text);
```

Example 9

Problem:

Write a program calculating the sum of first ***n*** natural numbers by using Abel equation:

$$s = n\,(n+1) / 2,$$

where:

s – sum of numbers, ***n*** – the last number in a sequence.

Discussion:

Input data: number ***n***

Output data: sum of numbers

Algorithm:

$$s = n\,(n+1) / 2$$


```
# include <stdio.h>
# include <stdlib.h>
   /* calculating the sum of first n natural numbers */
int main (void)
{
   char text [20];
   int    n,s;
   printf(" Enter the number n: ");
   gets(text);
   n = atoi(text);
   if (n!=0)
   {
   s = n * (n+1) /2;
   printf("The sum of first %d natural numbers equals: %d\n", n,s);
   getchar();
   } else printf(" Invalid data entered, try again\n");
   return 0;
}
```

Arithmetic expressions

- Order of operations executed in C is consistent with the rules of mathematics (parenthesis, then operator precedence and associativity).
- The arithmetic operator precedence from the highest to the lowest:
  - unary
  - multiplicative
  - additive
- The arithmetic operators have left-to-right associativity, i.e. operators with the same precedence are evaluated from left to right.

65

Arithmetic operators

C	Pascal	Operation
+	+	addition
-	-	subtraction
/	/ or div	division
*	*	multiplication
%	mod	modulo reduction - returns the remainder from integer division

66

do … while statement

do statement while (expression);

In *do…while* loop statement is executed, later on a logical value of expression is checked. If expression is true loop is executed once again.

Example:

```
int x =0;
do {
        x=x+1;
        printf("Number %d\n", x);
   } while (x<100);
```


Example 10

Problem:

Write the program reading the numbers from the keyboard and checking, if they are even or odd, until number *zero* will be entered.

Discussion:

Input: integer numbers for checking
Output: comment - *even* or *odd*

Algorithm:

1. Numbers should be read in a loop, until *zero* is entered.
2. For each number its parity is checked (% operator) and parity check result is printed.

```
/*File parity.c   (of integer numbers)*/

#include <stdio.h>
 int main()
{
   int n;
   printf("Program reads numbers and checks their parity\n");
   printf("until zero is entered\n");
   do
   {
     printf("Enter number: ");
     scanf("%d", &n);
     if((n%2)==0)
              printf("Number %d is even\n", n);
     else printf("Number %d is odd\n", n);
   } while (n!=0);
   getchar();
   return 0;
}
```


Example 11

Problem:

Write the program reading pairs of integer numbers ***a*** & ***b*** and checking, if ***a*** is divisible through ***b***. Numbers divisibility should be checked until the user wishes to continue. Correctness of the entered data should be checked.

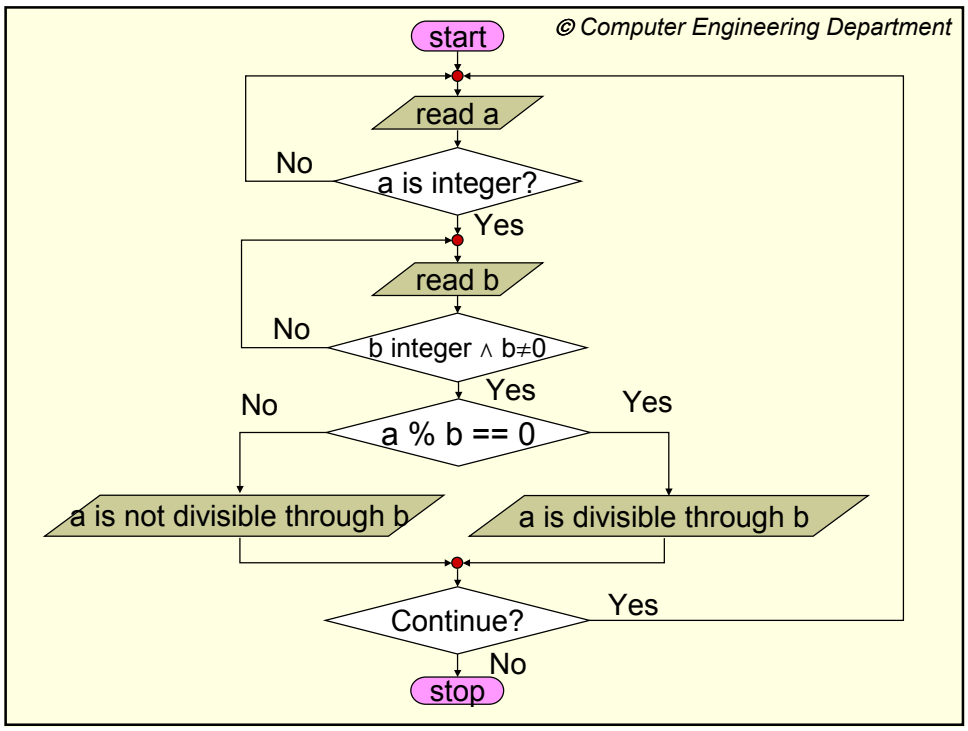
Discussion:

Use the ***atoi*** function.

Algorithm:

- Read the pairs of numbers ***a*** & ***b*** in a loop, check, if ***a*** is divisible through ***b*** and print the proper information on the screen. In the end check the loop condition.
- If the user wishes to stop and press „n" or „N" finish the program, in other case continue the loop.

70




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

int main() {
   int a, b;
   char z, tmp[10];
   printf("Program checks divisibility a through b\n");
   printf("Enter a pair of integer numbers \n");
   do {
    do {
      printf("Enter a: ");
      scanf("%s", &tmp);
    }while(atoi(tmp)==0);
     a = atoi(tmp);
    do {
      printf("Enter b: ");
      scanf("%s", &tmp);
    }while(atoi(tmp)==0);
     b = atoi(tmp);
     if((a%b)==0) printf(" Number a=%d is divisible through b=%d", a, b);
     else printf(" Number a=%d is not divisible through b=%d", a, b);
     printf("\nContinue? (n-finish)\n");
     z=getchar();
   }while((z!='n')&&(z!='N'));
   return 0;
}
```

72

Types of arithmetic expressions

- An expression type is determined by the types of its arguments. If types of different size are involved, the result will usually be of the larger size.
- In case of assignment, expression type is determined by the type of the assigned variable.

Examples:

```
int a,b;
double c,d;
b*c/d          // expression of double type
a = b*c/d      //expression of int type, b*c/d value is rounded
```


Type conversion and casting

- Casting forces type conversion, using cast operator ().
- The cast is made by putting the bracketed name of the required type just before the expression:

  (type_name) castet _expression

Examples:

```
5 / 7   //expression_result = 0 – integer division
(float)5 / 7   //expression_result = 0.71428
5 / 7.0        //expression_result = 0.71428
```

Increment and decrement operators

++	increment by 1
- -	decrement by 1

Increment and decrement operators may be **prefix** (before the variable: ++n) or **postfix** (after the variable: n++). They are different. In both cases n value is changed, but:

++n – increment n value before its use,

n++ - increment n value after its use.

Examples:

```
n=5;   x = n++; // results: x=5, n=6
       x = ++n; // results: x=6, n=6
```


Logical value of expression

Each expression in C has its logical value:

- zero value means „false”,
- non-zero value means „true”.

Variable of any type may be used as logical variable, e.g. integer variable.

Example:

```
int valid_data = 0; //logical value „false”
if (valid_data == 0) printf("Invalid data entered.\n");
```

Comparison operators

Relational operators:

>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

Equality operators:

==	equal to
!=	not equal to (different)

78

Logical operators

Operator	Function
!	negation (NOT)
&&	conjunction (AND)
||	disjunction (OR)

Comparison and logical operators return value 1 („true”) or 0 („false”) only.

79

Assignment operators and expressions

Expression:

expression1 op= expression2

/* where: $op \in \{+ - * / \%\}$ */

is equivalent to the expression:

expression1 = expression1 op expression2

Examples:

i +=2 or i = i + 2

x *= y + 1 or x = x * (y + 1)

xval % s [a+b] += 4

or xval % s [a+b] = xval % s [a+b] + 4

Example 12

- Find the smallest natural number *n*, which fulfils the following inequality:

$$1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}>\varepsilon \quad ,$$

where: ε is any positive number.

```c
# include <stdio.h>
# include <stdlib.h>
/* program finds the smallest natural number n, which
   fulfils the following inequality: 1+1/2+1/3+…+1/n>e */

int main (void)
{
char t[10];
int n = 1;
float e,s=0;
do {
    printf("Enter positive number e: ");
    gets(t);  e = atof(t);
} while (e <= 0);
    do   {
          s += 1/(float)n;  n++;
    } while (s <= e);
    printf("n= %d\n", n-1);
    getchar();
return (0);
}
```

84

Precedence and Associativity of C operators

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

85