Programming basics
(GKNB_INTA023)

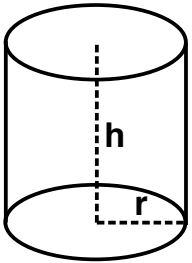
Hatwagner F. Miklós, PhD.

Széchenyi István University, Győr, Hungary

https://github.com/sze-info/ProgrammingBasics

October 8, 2020

Calculating the surface and volume of a cylinder



Tasks:

1. Read the height and radius of the cylinder
2. Calculate the surface and volume of the cylinder
   $V = r^2\pi h$
   $S = 2r\pi h + 2r^2\pi = 2r\pi(r + h)$

Type	Size	Number representation limits	Precision
float	4 bytes	$\pm 3,4 \cdot 10^{-38}$ – $\pm 3,4 \cdot 10^{+38}$	6-7 dec. digits
double	8 bytes	$\pm 1,7 \cdot 10^{-308}$ – $\pm 1,7 \cdot 10^{+308}$	15-16 dec. digits
long double	10 bytes	$\pm 1,2 \cdot 10^{-4932}$ – $\pm 1,2 \cdot 10^{+4932}$	19 dec. digits

Calculating the surface and volume of a cylinder

cylinder.c

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

int main(void) {
  double r, h;
  printf("Enter the radius of the cylinder: ");
  scanf("%lf", &r); // scanf: %lf -> double
  printf("Enter the height of the cylinder: ");
  scanf("%lf", &h); // printf: %f -> double
  printf("Volume: %f\n\tSurface: %f\n",
         r*r*M_PI*h, 2.*r*M_PI*(r+h));
  return 0;
}
```

Calculating the surface and volume of a cylinder

Main properties of floating point literals

- representation limits → `float.h`, eg.
  `DBL_MIN` the least positive normal number representable by type `double`
  `DBL_MAX` the greatest finite number that can be stored in a `double`
- the integer or the fractional part of the mantissa may be omitted, but not both of them!
- the decimal point or the exponent (`e`, `E`) part may be omitted, but not both of them!
- without any suffix the internal storage type is `double`

Calculating the surface and volume of a cylinder

Main properties of floating point literals, contd.

- Suffixes to change the internal storage type of a literal:
  - `f`, `F` (`float`)
  - `l`, `L` (`long double`)

Some floating point literals

-5., .3, 5.3, -5e4, 5.67E-12, -1.23e-4l, 5.F

Some of the (not necessarily standardized) literals of `math.h`

- `M_E` – Euler-constant
- `M_PI` – π
- `M_SQRT2` – $\sqrt{2}$

Calculating the surface and volume of a cylinder

Main properties of integer literals

- can be given in decimal, octal (`0...`) and hexadecimal (`0x...`, `0X...`) form
- suffixes to change the internal storage type:
  - `u`, `U` (unsigned)
  - `l`, `L` (long)

Integer variables and literals

```
int i = 1;                  unsigned ui = 8u;
int j = 010;  /* == 8 */    long li = 16L;
int k = 0x2A; /* == 42 */   unsigned long uli = 666Ul;
```

Main properties of integer literals, contd.

- representation limits of platform-dependent integer types → `limits.h`
- platform-independent, fixed size integer types, eg. `int32_t`, `uint16_t` → `stdint.h` (C99).

some details of `limits.h`

```
# define SCHAR_MIN (-128)
# define UCHAR_MAX 255
# define SHRT_MAX 32767
# define INT_MAX 2147483647
# define ULONG_MAX 18446744073709551615UL
```

Calculating absolute value

absolute1.c

```c
#include <stdio.h>

int main(void) {
  double v, abs;
  printf("Number: ");
  scanf("%lf", &v);
  printf("Absolute value: ");


  if(v < 0.) abs = -v;
  else abs = v;


  printf("%f\n", abs);
  return 0;
}
```

absolute2.c

```c
#include <stdio.h>

int main(void) {
  double v, abs;
  printf("Number: ");
  scanf("%lf", &v);
  printf("Absolute value: ");


  abs = v<0. ? -v : v;



  printf("%f\n", abs);
  return 0;
}
```

Calculating absolute value

Ternary, conditional operator (shorthand for if...else): ?:

if ... else

```
if(logicalExpression) {
    variable = valueIfTrue;
} else {
    variable = valueIfFalse;
}
```

Ternary operator

```
variable = logicalExpression ? valueIfTrue : valueIfFalse;
```

Triangle inequality

triangle5.c

```c
#include <stdio.h>
#include <stdbool.h>
#include <iso646.h>
#define SIDES 3

int main(void) {
  double sideArray[SIDES]; // side lengths can be racional numbers, too
  int i;
  bool valid = false;
  printf("Enter the sides of a triangle!\n");
  do {
    i = 0;
    while(i < SIDES) {
      do {
        printf("Length of side %c: ", 'A'+i);
        scanf("%lf", &sideArray[i]);
      } while(sideArray[i] <= 0.); // floating-point literal
      i++;
    }
    valid = (sideArray[0]+sideArray[1]>sideArray[2] and
             sideArray[1]+sideArray[2]>sideArray[0] and
             sideArray[2]+sideArray[0]>sideArray[1]);
    // ternary operator
    printf("The triangle is %s.\n", (valid ? "valid" : "invalid"));
  } while(not valid);
  return 0; }
```

Solving a quadratic equation

quadratic.c $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

```
1  #include <stdio.h>
2  #include <math.h> // sqrt() needs it
3
4  int main(void) {
5    double a, b, c;
6    printf("Solving equation ax^2+bx+c = 0\n"
7           "Enter the value of coefficient a: ");
8    scanf("%lf", &a);
9    if(a == 0.) {
10     printf("The equation is not quadratic!\n");
11   } else {
12     printf("Enter the value of coefficient b: "); scanf("%lf", &b);
13     printf("Enter the value of coefficient c: "); scanf("%lf", &c);
14     double d = b*b - 4.*a*c;
15     if(d < 0.) {
16       printf("The equation has no real root.\n");
17     } else {
18       printf("x1 = %f\nx2 = %f\n", (-b + sqrt(d)) / (2.*a),
19                                    (-b - sqrt(d)) / (2.*a));
20     }
21   }
22   return 0;
23 }
```

Solving a quadratic equation

Mathematical functions

- Standard function libraries → portability
- Header to be included: `math.h`
- GCC: linking of the floating-point library must be explicitely stated, eg.:
  `gcc -Wall -o quadratic quadratic.c -lm`
- The type of function parameters and return values are usually `double`
- Argument and return value of trigonometric functions are specified in radians

Solving a quadratic equation

Some often used mathematical function

Prototype	Goal
`double ceil(double x)`	returns the smallest integral value that is not less than x
`double cos(double x)`	cosine
`double cosh(double x)`	hyperbolic cosine
`double exp(double x)`	base-e exponential function
`double fabs(double x)`	absolute value of floating-point number
`double fmod(double x, double y)`	computes the floating-point remainder of dividing x by y
`double log(double x)`	natural logarithmic function
`double log10(double x)`	base-10 logarithmic function
`double pow(double x, double y)`	power function
`double sqrt(double x)`	square root

Fahrenheit – Celsius conversion

$$C = \frac{5}{9}(F - 32)$$

fahrCels1.c

```
1  #include <stdio.h>
2
3  int main(void) {
4     printf("Fahrenheit --> Celsius\n"
5            "Fahrenheit: ");
6     double f;
7     scanf("%lf", &f);
8     // Integer division, implicit type conversion
9     printf("Celsius: %f\n", (5/9)*(f-32));
10    return 0;
11 }
```

Output

```
Fahrenheit -> Celsius
Fahrenheit: 72
Celsius: 0.000000
```

Remarks:

- 5/9 → always 0!
- f-32 → implicit type conversion to double

Fahrenheit – Celsius conversion

Implicit/automatic type conversion: binary operators work with the same type of operands. In general, if types differ the smaller/more inaccurate operand is converted to the bigger/more accurate type.

one of the operands	the other operand
`long double`	*anything*→`long double`
`double`	*anything*→`double`
`float`	*anything*→`float`
integer promotion	*integer promotion*
`unsigned long`	*anything*→`unsigned long`
`long`→`(unsigned) long`	`unsigned int`→`(unsigned) long`
`long`	*anything*→`long`
`unsigned int`	*anything*→`unsigned int`
`int`	`int`

Integer promotion

Original type	Converted type	Conversion method
`char`	`int`	According to the default (signed/unsigned) char type.
`unsigned char`	`int`	Extension with zero-valued bits.
`signed char`	`int`	Extension with the value of the sign bit.
`short int`	`int`	Extension with the value of the sign bit.
`unsigned short`	`unsigned int`	Extension with zero-valued bits.

Attention!

- Conversions need time!
- A string is never converted to arithmetic value implicitely!

Fahrenheit – Celsius conversion

Explicit type conversion (Type casting)

fahrCels3.c

```
8   // Explicit, implicit type conv.
9   printf("Celsius: %f\n", ((double)5/9)*(f-32.));
```

Output

```
Fahrenheit --> Celsius
Fahrenheit: 72
Celsius: 22.222222
```

Fahrenheit – Celsius conversion

fahrCels2.c

```
8   // Implicit type conv.
9   printf("Celsius: %f\n", (5./9)*(f-32));
```

Output

```
Celsius: 22.222222
```

fahrCels3.c

```
8   // Explicit, implicit type conv.
9   printf("Celsius: %f\n", ((double)5/9)*(f-32.));
```

Output

```
Celsius: 22.222222
```

fahrCels4.c

```
8   // No conversion
9   printf("Celsius: %f\n", (5./9.)*(f-32.));
```

Output

```
Celsius: 22.222222
```

fahrCels5.c

```
8   // Pointless type casting
9   printf("Celsius: %f\n", (double)(5/9)*(f-32.));
```

Output

```
Celsius: 0.000000
```

Precedence and associativity of operators

Operator	Associativity
a++ a−−	left to right
fn() array[]	
++a −−a	right to left
+a −a	
!	
(type)	
*pointer	
&variable	
sizeof	
a*b a/b a%b	left to right
a+b a−b	
< <= > >=	
== !=	
&&	
||	
a?b:c	right to left
= += −= *= /= %=	
,	left to right

for loop

`for(`*<init-expression>*`;` *<repeat-expression>*`;` *<increment-expression>*`)` *statement*

1. evaluating *init-expression* if it is provided
2. executing *statement*, if the value of *repeat-expression* is true
3. evaluating *increment-expression* if it is provided, then go to 2

All *expressions* can be empty or compound using the comma operator. The empty *repeat-expression* evaluates to `true`. Usual scenario:

while

```
loopVariable = initialValue;
while(loopVariable < finalValue) {
    loopBody;
    loopVariable += step;
}
```

for

```
for(loopVariable=initialValue; loopVariable<finalValue; loopVariable += step) {
    loopBody;
}
```

Reading N numbers, storing and printing of them in reverse order

reverse1.c

```
5   int numbers[N], quantity=0;
6   while(quantity < N) {
7     printf("Number %d: ", quantity+1);
8     scanf("%d", &numbers[quantity]);
9     quantity++;
10  }
11  printf("\nIn reverse order:\n");
12  quantity = N-1;
13  while(quantity >= 0) {
14    printf("%d\t", numbers[quantity]);
15    quantity--;
16  }
```

reverse3.c

```
int numbers[N], quantity;
for(quantity=0; db<N; quantity++) {
  printf("Number %d: ", quantity+1);
  scanf("%d", &numbers[quantity]);
}

printf("\In reverse order:\n");
for(quantity=N-1; quantity>=0; quantity--) {
  printf("%d\t", numbers[quantity]);
}
```

for loop

General scenario:

while

```
statement1;
while(condition) {
    statement2;
    statement3;
}
```

for

```
for(statement1; condition; statement3) {
    statement2;
}
```

Converting a decimal number to binary number system

dectobin2.c

```
8   scanf("%d", &d);
9   i = 0;
10  while(d > 0) {
11    b[i] = d%2+'0'; d /= 2; i++;
12  }
```

dectobin3.c

```
8   for(scanf("%d", &d),i=0;
9       d>0;
10      d/=2,i++) {
11    b[i] = d%2+'0';
12  }
```

Mirroring a word in place

mirror1.c

```c
#include <stdio.h>
#include <string.h>
int main(void) {
   printf("Enter a word! ");
   char word[64];
   scanf("%s", word);
   int from, to;


   from = 0; to = strlen(word)-1;
   while(from < to) {
      char swap = word[from];
      word[from] = word[to];
      word[to] = swap;
      from++; to--;
   }


   printf("Mirrored: %s\n", word);
   return 0;
}
```

mirror2.c

```c
#include <stdio.h>
#include <string.h>
int main(void) {
   printf("Enter a word! ");
   char word[64];
   scanf("%s", word);
   int from, to;


   for(from=0, to=strlen(word)-1;
       from<to;
       from++, to--) {
      char swap = word[from];
      word[from] = word[to];
      word[to] = swap;
   }


   printf("Mirrored: %s\n", word);
   return 0;
}
```

for loop

fahrCels6.c

```
1  #include <stdio.h>
2  #define LOWER 0
3  #define UPPER 150
4  #define STEP 10
5
6  int main(void) {
7     printf("Fahrenheit\tCelsius\n"
8            "----------\t--------\n");
9     double f;
10
11    for(f=LOWER; f<=UPPER; f+=STEP)
12       printf("%f\t%f\n", f, (5./9.)*(f-32.));
13
14    return 0;
15 }
```

Output

```
Fahrenheit      Celsius
----------      --------
0.000000        -17.777778
10.000000       -12.222222
20.000000       -6.666667
30.000000       -1.111111
40.000000       4.444444
50.000000       10.000000
60.000000       15.555556
70.000000       21.111111
80.000000       26.666667
90.000000       32.222222
100.000000      37.777778
110.000000      43.333333
120.000000      48.888889
130.000000      54.444444
140.000000      60.000000
150.000000      65.555556
```