Programming basics
(GKNB_INTA023)

Hatwagner F. Miklós, PhD.

Széchenyi István University, Győr, Hungary

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

September 26, 2021

Triangle equality

triangle1.c ANSI C (C89) compliant implementation

```
1  #include <stdio.h>
2  int main(void) {
3    int a, b, c;
4    int valid = 0; // logical false
5    printf("Enter the sides of a triangle!\n");
6    do {
7      do { // start of the loop body
8        printf("Length of side A: ");
9        scanf("%d", &a);
10     } while(a <= 0); // end of the loop
11     do {
12       printf("Length of side B: ");
13       scanf("%d", &b);
14     } while(b <= 0);
15     do {
16       printf("Length of side C: ");
17       scanf("%d", &c);
18     } while(c <= 0);
19     if(a+b<=c || b+c<=a || c+a<=b) // or --> ||
20       printf("The triangle is invalid!\n");
21     else {
22       valid = 1; // logical true
23       printf("The triangle is valid.\n"); }
24   } while(!valid); // not --> !
25   return 0; }
```

Triangle equality

triangle2.c C99 compliant implementation; reading side lengths are repeated 3x!

```
1  #include <stdio.h>
2  #include <iso646.h>  // and, or, not
3  #include <stdbool.h> // bool
4  int main(void) {
5    int a, b, c;
6    bool valid = false; // more expressive type/value
7    printf("Enter the sides of a triangle!\n");
8    do {
9      do { // start of the loop body
10       printf("Length of side A: "); scanf("%d", &a);
11     } while(a <= 0); // end of the loop
12     do {
13       printf("Length of side B: "); scanf("%d", &b);
14     } while(b <= 0);
15     do {
16       printf("Length of side C: "); scanf("%d", &c);
17     } while(c <= 0);
18     if(a+b<=c or b+c<=a or c+a<=b) // or
19       printf("The triangle is invalid!\n");
20     else {
21       valid = true; // more expressive logical value
22       printf("The triangle is valid.\n"); }
23   } while(not valid); // not
24   return 0; }
```

Triangle equality

triangle3.c

```
1  #include <stdio.h>
2  #include <iso646.h>  // and, or, not
3  #include <stdbool.h> // bool
4  #define SIDES 3
5  int main(void) {
6    int sideArray[SIDES]; // 3 element array for storing side lengths
7    int i;                // index of the current side (0-2)
8    bool valid = false;
9    printf("Enter the sides of a triangle!\n");
10   do {
11     i = 0;
12     while(i < SIDES) { // The code snippet of reading side length appears only once
13       do {
14         printf("Length of the next side: "); scanf("%d", &sideArray[i]); // array indexing
15       } while(sideArray[i] <= 0);
16       i++;
17     }
18     if(sideArray[0]+sideArray[1]<=sideArray[2] or
19        sideArray[1]+sideArray[2]<=sideArray[0] or
20        sideArray[2]+sideArray[0]<=sideArray[1])
21       printf("The triangle is invalid!\n");
22     else {
23       valid = true; printf("The triangle is valid.\n"); }
24   } while(not valid);
25   return 0; }
```

Triangle equality

Array definition

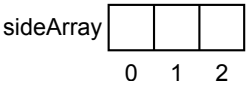
- *type name[size];*
- eg. `int sideArray[3];`
- *size* is a positive integer valued *constant expression*
- the value of a *constant expression* can be calculated compile-time

Memory requirement of an array

sizeof(name_of_the_array) $\equiv$ *size*sizeof(type)*

Accessing array elements

- *name[index]*
- $0 \leq index \leq size-1$



Triangle equality

triangle4.c

```c
#include <stdio.h>
#include <iso646.h>  // and, or, not
#include <stdbool.h> // bool
#define SIDES 3
int main(void) {
  int sideArray[SIDES];
  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: ", i+'A'); /* side's name */ scanf("%d", &sideArray[i]);
      } while(sideArray[i] <= 0);
      i++;
    }
    if(sideArray[0]+sideArray[1]<=sideArray[2] or
       sideArray[1]+sideArray[2]<=sideArray[0] or
       sideArray[2]+sideArray[0]<=sideArray[1])
      printf("The triangle is invalid!\n");
    else {
      valid = true; printf("The triangle is valid.\n"); }
  } while(not valid);
  return 0; }
```

Triangle equality

Generating side names

- ASCII codes of letters in ascending order are also increasing ('A' == 65, 'B' == 66, ..., 'Z' == 90)
- Digits are encoded similarly ('0' == 48, '1' == 49, ..., '9' == 57)
- Digit → ASCII code: `'0'+digit`
- ASCII code → digit: `character-'0'`
- Letters can be handled similarly

```
14        printf("Length of side %c: ", i+'A'); /* side's name */ scanf("%d", &sideArray[i]);
```

Format specifier: %c (single character, not terminated by zero!)
Character literals are between apostrophes!

Counting digits

counter1.c 1/2

```
1  #include <stdio.h>
2  int main(void) {
3    printf("Counting digits, whitespace and other characters\n"
4           "until EOF or Ctrl+D is given.\n\n");
5    int c, white=0, other=0;
6    int zero=0, one=0, two=0, three=0, four=0, // :(
7        five=0, six=0, seven=0, eight=0, nine=0;
8    while((c=getchar()) != EOF){
9      switch(c) {                          // Very ugly!
10       case '0': zero++; break;
11       case '1': one++; break;
12       case '2': two++; break;
13       case '3': three++; break;
14       case '4': four++; break;
15       case '5': five++; break;
16       case '6': six++; break;
17       case '7': seven++; break;
18       case '8': eight++; break;
19       case '9': nine++; break;
20       case ' ': case '\n': case '\t': white++; break;
21       default: other++; break;
22     }
23   }
```

Counting digits

counter1.c 2/2

```
24    printf("Digits:\n");
25    printf("Zeros:\t%d\n", zero); // Oh my God!
26    printf("Ones:\t%d\n", one);
27    printf("Twos:\t%d\n", two);
28    printf("Threes:\t%d\n", three);
29    printf("Fours:\t%d\n", four);
30    printf("Fives:\t%d\n", five);
31    printf("Sixs:\t%d\n", six);
32    printf("Sevens:\t%d\n", seven);
33    printf("Eights:\t%d\n", eight);
34    printf("Nines:\t%d\n", nine);
35    printf("White spaces: %d, others: %d\n", white, other);
36    return 0;
37  }
```

We urgently need an array!

Counting digits

counter2.c 1/2

```
1  #include <stdio.h>
2  #include <iso646.h>
3  #define PIECES 10
4  int main(void) {
5    printf("Counting digits, whitespace and other characters\n"
6           "until EOF or Ctrl+D is given.\n\n");
7    int i, c, white=0, other=0;
8    int digits[PIECES];  // 10 element array for the digits
9    i = 0;
10   while(i < PIECES) {
11     digits[i] = 0; // Zeroing the counters
12     i++;
13   }
```

Counting digits

counter2.c 2/2

```
15    while((c=getchar()) != EOF){
16      if(c>='0' and c<='9') {
17        i = c-'0';       // Converting a character (ASCII code) to a number,
18        digits[i]++;     // which is used as an index
19      } else if(c==' ' or c=='\n' or c=='\t') white++;
20      else other++;
21    }
22    printf("Digits:\n");
23    i = 0;               // Displaying the results
24    while(i < PIECES) {
25      printf("%d\t%d\n", i, digits[i]);
26      i++;
27    }
28    printf("White spaces: %d, others: %d\n", white, other);
29    return 0;
30  }
```

Counting digits

Array elements as counters

The quantity of digit i is stored at digits[i] (eg. 0 → digits[0], 1 → digits[1], etc.)

Initialization of arrays

- *type name[<size>]<=initializer_list>;*
- If the number of elements in *initializer_list* < *size* → remaining elements are reset to zero
- If the number of elements in *initializer_list* > *size* → ERROR!
- If *size* is not specified, the compiler counts the elements of *initializer_list*
- But at least one of *size* and *initializer_list* must exist!

Counting digits

counter3.c

```c
#include <stdio.h>
#include <iso646.h>
#define PIECES 10
int main(void) {
  printf("Counting digits, whitespace and other characters\n"
         "until EOF or Ctrl+D is given.\n\n");
  int c, i, white=0, other=0;
  int digits[PIECES] = {0}; // resetting counters to zero
  while((c=getchar()) != EOF){
    if(c>='0' and c<='9') ++digits[c-'0']; // increasing the counter
    else if(c==' ' or c=='\n' or c=='\t') ++white;
    else ++other;
  }
  printf("Digits:\n");
  i = 0;
  while(i < PIECES) {
    printf("%d\t%d\n", i, digits[i]);
    i++;
  }
  printf("White spaces: %d, others: %d\n", white, other);
  return 0;
}
```

Printing numbers in reverse order

reverse1.c

```c
#include <stdio.h>
#define N 5
int main(void) {
   printf("Enter %d numbers. The program prints them in reverse order.\n\n", N);
   int numbers[N], quantity=0;
   while(quantity < N) {
      printf("Number %d: ", quantity+1);
      scanf("%d", &numbers[quantity]);
      quantity++;
   }
   printf("\nIn reverse order:\n");
   quantity = N-1;
   while(quantity >= 0) {
      printf("%d\t", numbers[quantity]);
      quantity--;
   }
   printf("\n");
   return 0;
}
```

Printing numbers in reverse order

reverse2.c

```
1  #include <stdio.h>
2  #define N 5
3  int main(void) {
4     printf("Enter %d numbers. The program prints them in reverse order.\n\n", N);
5     int numbers[N], quantity=0;
6     while(quantity < N) {
7        printf("Number %d: ", quantity+1);
8        scanf("%d", &numbers[quantity++]);     // merger
9     }
10    printf("\nIn reverse order:\n");
11    while(quantity--) printf("%d\t", numbers[quantity]); // merger
12    printf("\n");
13    return 0;
14 }
```

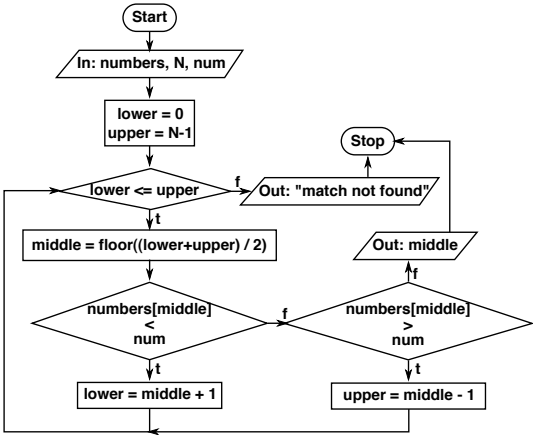
Linear (sequential, serial) search

sequential.c (→ Linear search)

```
1  #include <stdio.h>
2  #include <iso646.h>
3  #define N 10
4
5  int main(void) {
6    int numbers[N] = {13, -11, 0, 1, 42, 7, 14, 17, -23, 21};
7    printf("What are you looking for? ");
8    int i=0, num;
9    scanf("%d", &num);
10   while(i<N and numbers[i]!=num)
11     i++;
12   if(i == N)
13     printf("Match not found.\n");
14   else
15     printf("Match found at index %d.\n", i);
16   return 0;
17 }
```

Binary search

Binary search can be used only with ordered arrays!



numbers

-23	-11	0	1	7	13	14	17	21	42
0	1	2	3	4	5	6	7	8	9

num

1

	lower	middle	upper
1	0	?	9
2	0	4	9
3	0	4	3
4	0	1	3
5	2	1	3
6	2	2	3
7	3	2	3
8	3	3	3

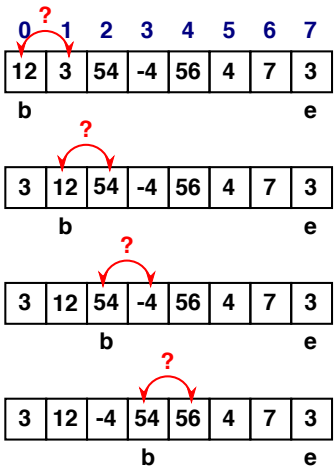
Binary search

binary.c

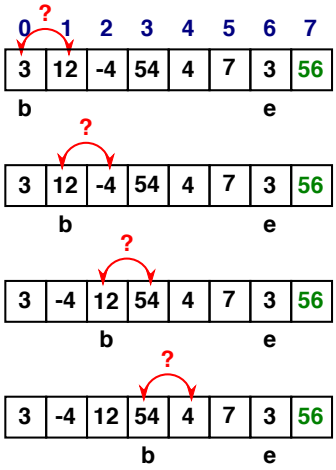
```c
#include <stdio.h>
#define N 10

int main(void) {
  int numbers[N] = {-23, -11, 0, 1, 7, 13, 14, 17, 21, 42};
  printf("What are you looking for? ");
  int num;
  scanf("%d", &num);
  int lower=0, upper=N-1, middle;
  while(lower <= upper) {
    middle = (lower+upper)/2;
    if(num < numbers[middle]) upper = middle-1;
    else if(num > numbers[middle]) lower = middle+1;
    else {
      printf("Match found at index %d.\n", middle);
      return 0;
    }
  }
  printf("Match not found, but it could be included at index %d.\n", lower);
  return 0;
}
```

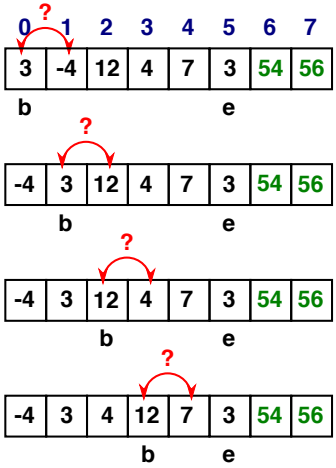
Bubble sort

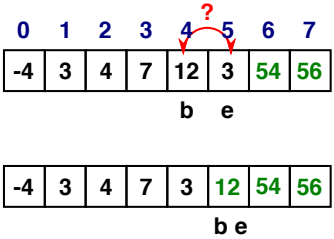

0 1 2 3 4 5 6 7
12 3 54 -4 56 4 7 3
b e
3 12 54 -4 56 4 7 3
b e
3 12 54 -4 56 4 7 3
b e
3 12 -4 54 56 4 7 3
b e
0 1 2 3 4 5 6 7
3 12 -4 54 56 4 7 3
b e
3 12 -4 54 4 56 7 3
b e
3 12 -4 54 4 7 56 3
b e
3 12 -4 54 4 7 3 56
b e

Bubble sort

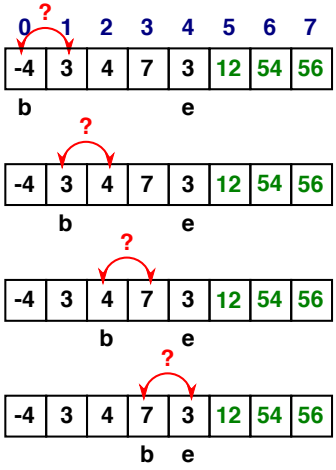

0 1 2 3 4 5 6 7
3 12 -4 54 4 7 3 56
b e
0 1 2 3 4 5 6 7
3 12 -4 54 4 7 3 56
b e
0 1 2 3 4 5 6 7
3 -4 12 54 4 7 3 56
b e
0 1 2 3 4 5 6 7
3 -4 12 54 4 7 3 56
b e
0 1 2 3 4 5 6 7
3 -4 12 4 54 7 3 56
b e
0 1 2 3 4 5 6 7
3 -4 12 4 7 54 3 56
b e
0 1 2 3 4 5 6 7
3 -4 12 4 7 3 54 56
b e

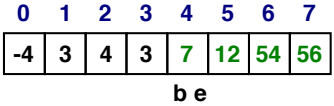
Bubble sort


0 1 2 3 4 5 6 7
3 -4 12 4 7 3 54 56
b e
-4 3 12 4 7 3 54 56
b e
-4 3 12 4 7 3 54 56
b e
-4 3 4 12 7 3 54 56
b e

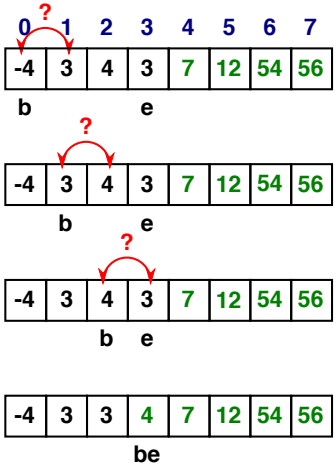

0 1 2 3 4 5 6 7
-4 3 4 7 12 3 54 56
b e
-4 3 4 7 3 12 54 56
b e

Bubble sort

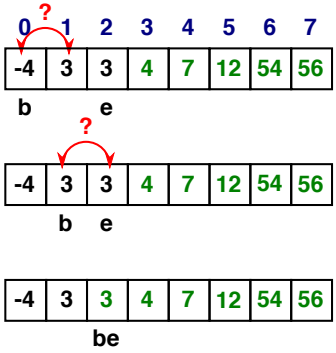

0 1 2 3 4 5 6 7
-4 3 4 7 3 12 54 56
b e
0 1 2 3 4 5 6 7
-4 3 4 7 3 12 54 56
b e
0 1 2 3 4 5 6 7
-4 3 4 7 3 12 54 56
b e
0 1 2 3 4 5 6 7
-4 3 4 7 3 12 54 56
b e


0 1 2 3 4 5 6 7
-4 3 4 3 7 12 54 56
b e

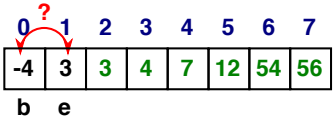
Bubble sort


0 1 2 3 4 5 6 7
?
-4 3 4 3 7 12 54 56
b e
?
-4 3 4 3 7 12 54 56
b e
?
-4 3 4 3 7 12 54 56
b e
-4 3 3 4 7 12 54 56
be

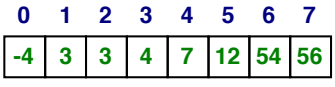
Bubble sort



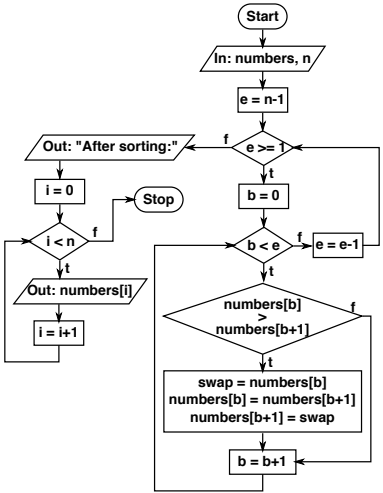
Bubble sort



Bubble sort



Bubble sort


Start
In: numbers, n
e = n-1
e >= 1
f
t
Out: "After sorting:"
i = 0
Stop
i < n
f
t
Out: numbers[i]
i = i+1
b = 0
b < e
f
t
e = e-1
numbers[b] > numbers[b+1]
f
t
swap = numbers[b]
numbers[b] = numbers[b+1]
numbers[b+1] = swap
b = b+1

Bubble sort

bubble.c (→ Bubble sort)

```c
#include <stdio.h>
int main(void) {
  int numbers[] = {12, 3, 54, -4, 56, 4, 7, 3};
  int n = sizeof(numbers)/sizeof(numbers[0]); // Calculating array size
  int e=n-1, b;
  while(e>=1) {
    b = 0;
    while(b<e) {
      if(numbers[b]>numbers[b+1]) {
        int swap = numbers[b];
        numbers[b] = numbers[b+1];
        numbers[b+1] = swap;
      }
      b++;
    }
    e--;
  }
  int i = 0;
  printf("After sorting:\n");
  while(i < n) {
    printf("%d\t", numbers[i]);
    i++;
  }
  printf("\n");
  return 0;
}
```

Basics of string handling

No type in C for strings! → character arrays terminated by the null character (`'\0'`)

	0	1	2	3	4
s	J	a	n	e	'\0'

String manipulation with functions, eg.

`strcat`
: Concatenates (appends) the content of the second string with the first

`strcpy`
: Copies the content of the second string in the first one

`strlen`
: Determines the length of the string (without the `'\0'` character)

`strcmp`
: Compares the content of strings (based on ASCII codes)

Required header: `string.h`

Basics of string handling

string.c

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

int main(void) {
  char s1[128] = "Tom"; // The terminating '\0' is placed at
  char s2[] = "Jerry";  // the end of the string automatically
  strcat(s1, " and ");  // conCATenation
  strcat(s1, s2);       // Is s1 able to store all the characters?
                        // Format specifier: %s
  printf("Title of the tale: %s\n", s1);
                        // LENgth; z -> size_t, u -> unsigned
  printf("Title length: %zu\n", strlen(s1));
                        // l -> long, u -> unsigned
  printf("Memory needed: %lu bytes.\n", sizeof(s1));
  strcpy(s1, "The Flinstones"); // CoPY
  printf("Another tale: %s\n", s1);
```

Basics of string handling

string.c

```
17    s1[3] = '\0'; s2[0] = 'G'; // Any characters can be modified
18    printf("Not funny: %s %s\n", s1, s2);
19                                // CoMParing the contents
20    int comp = strcmp(s1, s2);
21    if(comp < 0) {
22      printf("%s in front of %s.\n", s1, s2);
23    } else if(comp > 0) {
24      printf("%s follows %s.\n", s1, s2);
25    } else {
26      printf("%s and %s are the same.\n", s1, s2);
27    }
28    return 0;
29  }
```

Output

```
Title of the tale: Tom and Jerry
Title length: 13
Memory needed: 128 bytes.
Another tale: The Flinstones
Not funny: The Gerry
The follows Gerry.
```

Converting binary numbers to decimals

bintodec.c

```c
#include <stdio.h>

int main(void) {
  char b[64];
  unsigned d, i;
  printf("Enter a binary number!\n");
  scanf("%s", b); // Format specifier: %s, no address-of operator!!!
  d = i = 0;
  while(b[i] != '\0') { // Please, do not use strlen!
    d = d*2 + b[i]-'0'; // In general, it is VERY SLOW
    i++;
  }
  printf("In decimal number system: %d\n", d);
  return 0;
}
```

Converting decimal numbers to binary

dectobin.c

```c
#include <stdio.h>
int main(void) {
  char b[100];
  int d, i;
  printf("Enter a number in decimal number system!\n");
  scanf("%d", &d);
  i = 0;
  while(d > 0) {
    b[i] = d%2+'0'; d /= 2; i++;
  }
  printf("In binary number system: ");
  i--;
  while(i >= 0) {
    printf("%c", b[i]); // Printing a single character
    i--;
  }
  printf("\n");
  return 0;
}
```

Checking Neptun code

neptun1.c

```c
#include <stdio.h>
#include <stdbool.h>
#include <iso646.h>
#include <string.h>
int main(void) {
  bool invalid;
  char neptun[64]; // Enough space for too long codes
  do {
    invalid = false;
    printf("Enter your Neptun code: "); scanf("%s", neptun);
    if(strlen(neptun) != 6) { // checking length
      printf("It must contain six characters!\n"); invalid = true;
    } else {
      unsigned i=0;
      while(not invalid and neptun[i]!='\0') {
        char c = neptun[i];
        bool digit = c>='0' and c<='9';
        bool upper = c>='A' and c<='Z';
        bool lower = c>='a' and c<='z';
        if(not digit and not upper and not lower) {
          printf("Only alphanumeric characters are allowed!\n");
          invalid = true; }
        i++; } }
  } while(invalid);
  printf("The code is valid.\n");
  return 0; }
```

Checking Neptun code

neptun2.c

```c
#include <stdio.h>
#include <ctype.h> // toupper() needs it
#include <stdbool.h>
#include <iso646.h>
#include <string.h>
int main(void) {
  bool invalid;
  char neptun[64];
  do {
    invalid = false;
    printf("Enter your Neptun code: "); scanf("%s", neptun);
    if(strlen(neptun) != 6) {
      printf("It must contain six characters!\n"); invalid = true;
    } else {
      unsigned i=0;
      // ASCII code -> integer; 0 -> false, everything else -> true
      while(not invalid and neptun[i]) {
        char c = toupper(neptun[i]); // converting to uppercase letter
        if((c<'0' or c>'9') and (c<'A' or c>'Z')) {
          printf("Only alphanumeric characters are allowed!\n");
          invalid = true; }
        i++; } }
  } while(invalid);
  printf("The code is valid.\n");
  return 0; }
```

Checking Neptun code

Classification and conversion of characters

- `ctype.h` must be included
- May be implemented by functions or macros (preprocessor)
- The type of parameter is `int`, but the values must be representable by an `unsigned char` or `EOF`
- The return value is `int`, consider as logical value

Fn./macro name	Goal
`islower(c)`	is c a lowercase letter?
`isupper(c)`	is c an uppercase letter?
`isalpha(c)`	is c a letter?
`isdigit(c)`	is c a digit?
`isalnum(c)`	is c alphanumeric?
`isxdigit(c)`	is c a hexadecimal digit?
`isspace(c)`	is c whitespace?
`isprint(c)`	canc be printed?
`tolower(c)`	lowercase version of c if c is an uppercase letter
`toupper(c)`	uppercase version of c if c is a lowercase letter

Checking Neptun code

neptun3.c

```
1  #include <stdio.h>
2  #include <ctype.h> // isalnum() needs it
3  #include <stdbool.h>
4  #include <iso646.h>
5  #include <string.h>
6  int main(void) {
7    bool invalid;
8    char neptun[64];
9    do {
10     invalid = false;
11     printf("Enter your Neptun code: "); scanf("%s", neptun);
12     if(strlen(neptun) != 6) {
13       printf("It must contain six characters!\n"); invalid = true;
14     } else {
15       unsigned i=0;
16       while(not invalid and neptun[i]) {
17         if(not isalnum(neptun[i])) { // is it an alphanumeric character?
18           printf("Only alphanumeric characters are allowed!\n");
19           invalid = true; }
20         i++; } }
21   } while(invalid);
22   printf("The code is valid.\n");
23   return 0; }
```