Programming basics
(GKNB_INTA023)

Hatwagner F. Miklós, PhD.

Széchenyi István University, Győr, Hungary

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

September 22, 2020

Triangle equality

triangle1.c ANSI C (C89) compliant implementation

```c
#include <stdio.h>
int main(void) {
  int a, b, c;
  int valid = 0; // logical false
  printf("Enter the sides of a triangle!\n");
  do {
    do { // start of the loop body
      printf("Length of side A: ");
      scanf("%d", &a);
    } while(a <= 0); // end of the loop
    do {
      printf("Length of side B: ");
      scanf("%d", &b);
    } while(b <= 0);
    do {
      printf("Length of side C: ");
      scanf("%d", &c);
    } while(c <= 0);
    if(a+b<=c || b+c<=a || c+a<=b) // or --> ||
      printf("The triangle is invalid!\n");
    else {
      valid = 1; // logical true
      printf("The triangle is valid.\n"); }
  } while(!valid); // not --> !
  return 0; }
```

Triangle equality

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

```c
#include <stdio.h>
#include <iso646.h>  // and, or, not
#include <stdbool.h> // bool
int main(void) {
  int a, b, c;
  bool valid = false; // more expressive type/value
  printf("Enter the sides of a triangle!\n");
  do {
    do { // start of the loop body
      printf("Length of side A: "); scanf("%d", &a);
    } while(a <= 0); // end of the loop
    do {
      printf("Length of side B: "); scanf("%d", &b);
    } while(b <= 0);
    do {
      printf("Length of side C: "); scanf("%d", &c);
    } while(c <= 0);
    if(a+b<=c or b+c<=a or c+a<=b) // or
      printf("The triangle is invalid!\n");
    else {
      valid = true; // more expressive logical value
      printf("The triangle is valid.\n"); }
  } while(not valid); // not
  return 0; }
```

Triangle equality

triangle3.c

```c
#include <stdio.h>
#include <iso646.h>  // and, or, not
#include <stdbool.h> // bool
#define SIDES 3
int main(void) {
  int sideArray[SIDES]; // 3 element array for storing side lengths
  int i;                // index of the current side (0-2)
  bool valid = false;
  printf("Enter the sides of a triangle!\n");
  do {
    i = 0;
    while(i < SIDES) { // The code snippet of reading side length appears only once
      do {
        printf("Length of the next side: "); scanf("%d", &sideArray[i]); // array indexing
      } 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

Array definition

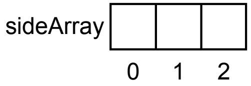
- *type name[size];*
- eg. `int sideArray[3];`
- *size* is a positive integer valued *constant expression*
- the the value of a *constant expression* can be calculated compile-time

Memory requirement of an array

$$\textit{sizeof(name\_of\_the\_array)} \equiv \textit{size*sizeof(type)}$$

Accessing array elements

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



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

```
    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

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

Counting digits

counter1.c 2/2

```c
    printf("Digits:\n");
    printf("Zeros:\t%d\n", zero); // Oh my God!
    printf("Ones:\t%d\n", one);
    printf("Twos:\t%d\n", two);
    printf("Threes:\t%d\n", three);
    printf("Fours:\t%d\n", four);
    printf("Fives:\t%d\n", five);
    printf("Sixs:\t%d\n", six);
    printf("Sevens:\t%d\n", seven);
    printf("Eights:\t%d\n", eight);
    printf("Nines:\t%d\n", nine);
    printf("White spaces: %d, others: %d\n", white, other);
    return 0;
}
```

We urgently need an array!

Counting digits

counter2.c 1/2

```
#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 i, c, white=0, other=0;
  int digits[PIECES];  // 10 element array for the digits
  i = 0;
  while(i < PIECES) {
    digits[i] = 0; // Zeroing the counters
    i++;
  }
```

Counting digits

counter2.c 2/2

```
  while((c=getchar()) != EOF){
    if(c>='0' and c<='9') {
      i = c-'0';         // Converting a character (ASCII code) to a number,
      digits[i]++;       // which is used as an index
    } else if(c==' ' or c=='\n' or c=='\t') white++;
    else other++;
  }
  printf("Digits:\n");
  i = 0;                 // Displaying the results
  while(i < PIECES) {
    printf("%d\t%d\n", i, digits[i]);
    i++;
  }
  printf("White spaces: %d, others: %d\n", white, other);
  return 0;
}
```

Counting digits

Array elements as counters

The number of digit i is stored at digits[i] (eg. 0 $\rightarrow$ digits[0], 1 $\rightarrow$ digits[1], etc.)

Initialization of arrays

- *type name[<size>]<=initializer_list>;*
- If the number of elements in *initializer_list* $<$ *size* $\rightarrow$ remaining elements are reset to zero
- If the number of elements in *initializer_list* $>$ *size* $\rightarrow$ 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!