

Programming basics

(GKNB_INTA023)

Hatwagner F. Miklós, PhD.

Széchenyi István University, Győr, Hungary

September 22, 2020

Minimum and maximum search

minmax1.c

```
1  #include <stdio.h>
2
3  int main(void) {
4      printf("Type non-negative integers and\n"
5             "the program finds their minimum and maximum.\n"
6             "Exit by entering a negative number.\n");
7      int quantity=0, current=1; // initialization
8      int min, max;
9      while(current >= 0) {
10         printf("Next number: ");
11         scanf("%d", &current);
12         if(current >= 0) {
13             if(quantity == 0) min = max = current; // multiple assignment
14             else if(current > max) max = current;
15             else if(current < min) min = current;
16             quantity++; // increase by one
17         }
18     }
19     if(quantity > 0) printf("Minimum: %d\nMaximum: %d\n",
20                             min, max);
21     else printf("You did not enter any numbers.\n");
22     return 0;
23 }
```

Minimum and maximum search

Variable

declaration providing type and name, place: before usage (C99)

definition declaration + memory allocation

initialization specifying the initial value at definition, eg. `int current=0;`

operator = (assignment)

- associativity: right to left
- `min = max = current;` \equiv `max = current; min = max;`

Multi-directional branching: *if(...) ... else if(...) ... else if(...) ... else ...*

Minimum and maximum search

Increment and decrement operators

`++` increase the value of a variable by one

`--` decrease the value of a variable by one

Prefix and postfix usage → order (precedence) of operations!

The effect of pre/postfix operator usage on the result

```
int a, b; // the values of a and b are undefined
b = 6;    // the value of b is 6 from now on
a = ++b;  // 1) b increases to 7
          // 2) and it is assigned to variable a
a = b++;  // 1) the value of b is assigned to a again; ineffective
          // 2) b increases to 8
```

Minimum and maximum search

minmax2.c

```
1  #include <stdio.h>
2
3  int main(void) {
4      printf("Type non-negative integers and\n"
5             "the program finds their minimum and maximum.\n"
6             "Exit by entering a negative number.\n");
7      int quantity=0, current; // current does not need initialization
8      int min, max;
9      printf("Next number: "); // first occurrence of the code snippet
10     scanf("%d", &current);
11     while(current >= 0) {
12         if(quantity == 0) min = max = current; // a selection is missing from here
13         else if(current > max) max = current;
14         else if(current < min) min = current;
15         quantity++;
16         printf("Next number: "); // second occurrence of the code snippet
17         scanf("%d", &current);
18     }
19     if(quantity > 0) printf("Minimum: %d\nMaximum: %d\n", min, max);
20     else printf("You did not enter any numbers.\n");
21     return 0;
22 }
```

Reading a positive number

positive1.c

```
1  #include <stdio.h>
2
3  int main(void) {
4      int number;
5      printf("Enter a positive number! ");    // message 1
6      scanf("%d", &number);                  // reading 1
7      while(number <= 0) { // repeat if input is erroneous
8          printf("Enter a positive number! "); // message 2
9          scanf("%d", &number);                // reading 2
10     }
11     printf("Read value: %d\n", number);
12     return 0;
13 }
```

Reading a positive number

positive2.c

```
1  #include <stdio.h>
2
3  int main(void) {
4      int number = -1;    // the initialization forces
5      while(number <= 0) { // the execution of loop body
6          printf("Enter a positive number! ");
7          scanf("%d", &number);
8      }
9      printf("Read value: %d\n", number);
10     return 0;
11 }
```

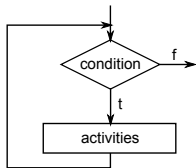
Reading a positive number

positive3.c

```
1  #include <stdio.h>
2
3  int main(void) {
4      int number; // no need to initialize it to
5      do {        // execute the loop body at least once
6          printf("Enter a positive number! ");
7          scanf("%d", &number);
8      } while(number <= 0);
9      printf("Read value: %d\n", number);
10     return 0;
11 }
```

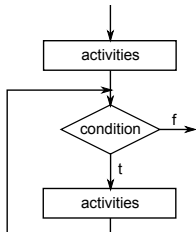

Triangle inequality

Testing the condition of a loop at the end of the body \rightarrow it is executed at least once



```
do {  
    activities  
} while(conditional expression);
```

Forcing the execution of the loop body in case of a *while* loop



```
activities  
while(conditional expression) {  
    activities  
}
```

Triangle inequality

triangle1.c

```
1 #include <stdio.h>
2 #include <stdbool.h>
3 int main(void) {
4     int a, b, c;
5     bool valid = false; // boolean (logical) type
6     printf("Enter the sides of a triangle!\n");
7     do {
8         do { // start of the loop body
9             printf("Length of side A: ");
10            scanf("%d", &a);
11        } while(a <= 0); // end of the loop
12        do {
13            printf("Length of side B: ");
14            scanf("%d", &b);
15        } while(b <= 0);
16        do {
17            printf("Length of side C: ");
18            scanf("%d", &c);
19        } while(c <= 0);
20        if(a+b<=c || b+c<=a || c+a<=b) // logical OR
21            printf("The triangle is invalid!\n");
22        else {
23            valid = true;
24            printf("The triangle is valid.\n"); }
25    } while(!valid);
26    return 0; }
```

Triangle inequality

Logical operators

- `!:` logical *not*
- `||:` logical (permissive) *or*
- `&&:` logical *and*

Logical (boolean) type: `bool`

Allowed values: `true`, `false`

Criterion of usage: `stdbool.h` (C99)

Truth table

a	b	!a	a b	a && b
false	false	true	false	false
false	true	true	true	false
true	false	false	true	false
true	true	false	true	true

Optimization of (sub)expressions: short-circuit evaluation

Triangle inequality

triangle2.c

```
1  #include <stdio.h>
2  int main(void) {
3      int a, b, c;
4      printf("Enter the sides of a triangle in ascending order!\n");
5      do {
6          printf("Length of side A: ");
7          scanf("%d", &a);
8      } while(a <= 0);
9      do {
10         printf("Length of side B: ");
11         scanf("%d", &b);
12     } while(b < a);
13     do {
14         printf("Length of side C: ");
15         scanf("%d", &c);
16     } while(c < b || a+b <= c);
17     return 0;
18 }
```

Triangle inequality

triangle3.c

```
1 #include <stdio.h>
2 #include <iso646.h> // and, or, not
3 int main(void) {
4     int a, b, c;
5     printf("Enter the sides of a triangle in ascending order!\n");
6     do {
7         printf("Length of side A: ");
8         scanf("%d", &a);
9     } while(a <= 0);
10    do {
11        printf("Length of side B: ");
12        scanf("%d", &b);
13    } while(b < a);
14    do {
15        printf("Length of side C: ");
16        scanf("%d", &c);
17    } while(c < b or a+b <= c); // it could be || as well
18    return 0;
19 }
```

Triangle inequality

More expressive syntax of logical operators: `iso646.h`

- `not`: logical *not* (\equiv `!`)
- `or`: logical (permissive) *or* (\equiv `||`)
- `and`: logical *and* (\equiv `&&`)

Drawing a circular plate

circle1.c

```
1  #include <stdio.h>
2
3  int main(void) {
4      int row = -5; // The radius of the circle is 5
5      while(row <= 5) {
6          int column = -5;
7          while(column <= 5) {
8              if(5*5 >= row*row + column*column) printf("*");
9              else printf(" ");
10             column++;
11         }
12         row++;
13         printf("\n");
14     }
15     return 0;
16 }
```

Drawing a circular plate

Output

```

      *
    **
  ***
****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****

```

Problems:

- limited cursor positioning capabilities
- The same constant (radius, 5) occurs at various places: hard (slow) to modify, error prone
- Characters are approx. twice as high as wide \rightarrow ellipse

Drawing a circular plate

circle2.c

```
1  #include <stdio.h>
2  #define R 10 // The radius of the circle is now 10
3
4  int main(void) {
5      int row = -R;
6      while (row <= R) {
7          int column = -R;
8          while (column <= R) {
9              if (R*R >= row*row + column*column) printf("*");
10             else printf(" ");
11             column++;
12         }
13         row += 2; // increase by two
14         printf("\n");
15     }
16     return 0;
17 }
```

Drawing a circular plate

#define (preprocessor directive)

- alias name for a literal, simple *macro*
- the preprocessor replaces the occurrences of the macro with the literal (except inside string literals, keywords, ... it's smart enough)
- **No semicolon at the end!**

Compound assignment operators

- `row += 2;` \equiv `row = row+2;`
- `+=`, `-=`, `*=`, `/=`, `%=`

Unary `+` and `-` operators

Output

```
      *
    *****
    *****
    *****
    *****
*****
*****
*****
*****
*****
      *
```

Counting characters, words and lines

counter1.c

```
1  #include <stdio.h>
2  #include <stdbool.h>
3
4  int main(void) {
5      int c, numLines, numWords, numChars;
6      bool insideWord = false;
7      printf("Counting the characters, words and lines of the input\n"
8             "Exit: Ctrl+D or EOF.\n\n");
9      numLines = numWords = numChars = 0;
10     while((c=getchar()) != EOF){
11         ++numChars;
12         if(c == '\n') ++numLines;
13         if(c==' ' || c=='\n' || c=='\t') insideWord = false;
14         else if(!insideWord){
15             insideWord = true;
16             ++numWords;
17         }
18     }
19     printf("Number of lines = %d, words = %d, characters = %d\n",
20           numLines, numWords, numChars);
21     return 0;
22 }
```

Counting characters, words and lines

Reading exactly one character: `int getchar(void);`

Characters are stored with `int` type

End Of File (input): `EOF`

Required header: `<stdio.h>`

Watch out for the operator precedence! `while((c=getchar()) != EOF){`

The role of `insideWord`

Operator precedence and associativity

Operator	Associativity
$a++$ $a--$	left to right
$++a$ $--a$	
$+a$ $-a$	right to left
$!$	
$sizeof$	
$a*b$ a/b $a\%b$	
$a+b$ $a-b$	
$<$ $<=$ $>$ $>=$	left to right
$==$ $!=$	
$\&\&$	
$ $	
$=$ $+=$ $-=$ $*=$ $/=$ $\% =$	right to left
$,$	left to right

ordinal1.c

```
1 #include <stdio.h>
2 int main(void) {
3     printf("Number: ");
4     int number;
5     scanf("%d", &number);
6     if(number == 0) printf("0");
7     else {
8         printf("%d", number);
9         if(number > 10 && number < 21) printf("th");
10        else if(number % 10 == 1) printf("st");
11        else if(number % 10 == 2) printf("nd");
12        else if(number % 10 == 3) printf("rd");
13        else printf("th");
14    }
15    return 0;
16 }
```

Problems: several branches, unnecessary divisions

ordinal2.c

```
1  #include <stdio.h>
2  int main(void) {
3      printf("Number: ");
4      int number;
5      scanf("%d", &number);
6      if(number == 0) printf("0");
7      else {
8          printf("%d", number);
9          if(number > 10 && number < 21) printf("th");
10         else switch(number % 10) {
11             case 1: printf("st"); break;
12             case 2: printf("nd"); break;
13             case 3: printf("rd"); break;
14             default: printf("th");
15         }
16     }
17     return 0;
18 }
```

- `switch(expression) statement`
- `expression` is int type
- `statement` can contain
 - multiple `case constant-expression: statements`,
 - zero or more `default: statements`
- execution of `switch` stops at
 - the end of the block of `switch`
 - the first `break` statement
- `constant-expression` is represented by type int
- comparison of `expression` and `constant-expression`
- more `case` labels may belong to the same statement, but all labels must be unique
- `switch` statements may be nested

Minimum and maximum search

minmax2.c (Reminder)

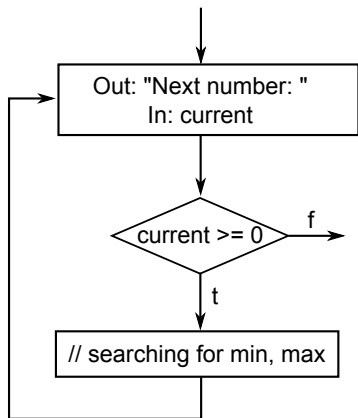
```
1  #include <stdio.h>
2
3  int main(void) {
4      printf("Type non-negative integers and\n"
5             "the program finds their minimum and maximum.\n"
6             "Exit by entering a negative number.\n");
7      int quantity=0, current; // current does not need initialization
8      int min, max;
9      printf("Next number: "); // first occurrence of the code snippet
10     scanf("%d", &current);
11     while(current >= 0) {
12         if(quantity == 0) min = max = current; // a selection is missing from here
13         else if(current > max) max = current;
14         else if(current < min) min = current;
15         quantity++;
16         printf("Next number: "); // second occurrence of the code snippet
17         scanf("%d", &current);
18     }
19     if(quantity > 0) printf("Minimum: %d\nMaximum: %d\n", min, max);
20     else printf("You did not enter any numbers.\n");
21     return 0;
22 }
```

Minimum and maximum search

minmax3.c

```
1  #include <stdio.h>
2  int main(void) {
3      printf("Type non-negative integers and\n"
4             "the program finds their minimum and maximum.\n"
5             "Exit by entering a negative number.\n");
6      int quantity=0, current;
7      int min, max;
8      // operator ,
9      while(printf("Next number: "), scanf("%d", &current), current >= 0) {
10         if(!quantity) min = max = current; // operator !
11         else if(current > max) max = current;
12         else if(current < min) min = current;
13         quantity++;
14     }
15     // logical expression
16     if(quantity) printf("Minimum: %d\nMaximum: %d\n", min, max);
17     else printf("You did not enter any numbers.\n");
18     return 0;
19 }
```

Minimum and maximum search



Operator ,

- makes possible to use a compound statement of multiple statements where only a single statement is allowed
- value of the expression = value of the last subexpression

Logical expressions

- bool type vs. *integer* types
- `false` \equiv 0
- `true` \equiv 1
- zero \rightarrow false
- non-zero values \rightarrow true
- `if(quantity) ...` \equiv `if(quantity != 0) ...`
- `if(!quantity) ...` \equiv `if(quantity == 0) ...`