Programming basics
(GKNB_INTA023)

Hatwagner F. Miklós, PhD.

Széchenyi István University, Győr, Hungary

September 20, 2020

Minimum and maximum search

minmax1.c

```c
#include <stdio.h>

int main(void) {
  printf("Type non-negative integers and\n"
         "the program finds their minimum and maximum.\n"
         "Exit by entering a negative number.\n");
  int quantity=0, current=1; // initialization
  int min, max;
  while(current >= 0) {
    printf("Next number: ");
    scanf("%d", &current);
    if(current >= 0) {
      if(quantity == 0) min = max = current; // multiple assignment
      else if(current > max) max = current;
      else if(current < min) min = current;
      quantity++; // increase by one
    }
  }
  if(quantity > 0) printf("Minimum: %d\nMaximum: %d\n",
    min, max);
  else printf("You did not enter any numbers.\n");
  return 0;
}
```

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;` ≡ `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 $\rightarrow$ 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

```c
#include <stdio.h>

int main(void) {
   printf("Type non-negative integers and\n"
          "the program finds their minimum and maximum.\n"
          "Exit by entering a negative number.\n");
   int quantity=0, current; // current does not need initialization
   int min, max;
   printf("Next number: "); // first occurrence of the code snippet
   scanf("%d", &current);
   while(current >= 0) {
      if(quantity == 0) min = max = current; // a selection is missing from here
      else if(current > max) max = current;
      else if(current < min) min = current;
      quantity++;
      printf("Next number: "); // second occurrence of the code snippet
      scanf("%d", &current);
   }
   if(quantity > 0) printf("Minimum: %d\nMaximum: %d\n", min, max);
   else printf("You did not enter any numbers.\n");
   return 0;
}
```

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 }
```

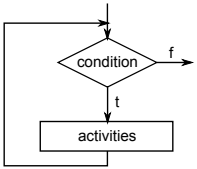
Reading a positive number

positive3.c

```c
#include <stdio.h>

int main(void) {
  int number; // no need to initialize it to
  do {        // execute the loop body at least once
    printf("Enter a positive number! ");
    scanf("%d", &number);
  } while(number <= 0);
  printf("Read value: %d\n", number);
  return 0;
}
```

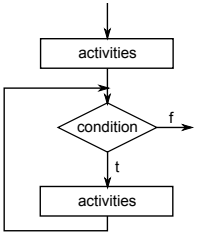
Triangle inequality

Testing the condition of a loop at the end of the body → 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

```c
#include <stdio.h>
#include <stdbool.h>
int main(void) {
  int a, b, c;
  bool valid = false; // logikai tipus
  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) // logical OR
      printf("The triangle is invalid!\n");
    else {
      valid = true;
      printf("The triangle is valid.\n"); }
  } while(!valid);
  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

```c
#include <stdio.h>
int main(void) {
   int a, b, c;
   printf("Enter the sides of a triangle in ascending order!\n");
   do {
      printf("Length of side A: ");
      scanf("%d", &a);
   } while(a <= 0);
   do {
      printf("Length of side B: ");
      scanf("%d", &b);
   } while(b < a);
   do {
      printf("Length of side C: ");
      scanf("%d", &c);
   } while(c<b || a+b<=c);
   return 0;
}
```

Triangle inequality

triangle3.c

```c
#include <stdio.h>
#include <iso646.h> // and, or, not
int main(void) {
  int a, b, c;
  printf("Enter the sides of a triangle in ascending order!\n");
  do {
    printf("Length of side A: ");
    scanf("%d", &a);
  } while(a <= 0);
  do {
    printf("Length of side B: ");
    scanf("%d", &b);
  } while(b < a);
  do {
    printf("Length of side C: ");
    scanf("%d", &c);
  } while(c<b or a+b<=c); // it could be || as well
  return 0;
}
```

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

```c
#include <stdio.h>

int main(void) {
  int row = -5; // The radius of the circle is 5
  while(row <= 5) {
    int column = -5;
    while(column <= 5) {
      if(5*5 >= row*row + column*column) printf("*");
      else printf(" ");
      column++;
    }
    row++;
    printf("\n");
  }
  return 0;
}
```

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;` ≡ `row = row+2;`
- +=, −=, *=, /=, %=

Unary + and − operators

Output

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

Counting characters, words and lines

counter1.c

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

int main(void) {
  int c, numLines, numWords, numChars;
  bool insideWord = false;
  printf("Counting the characters, words and lines of the input\n"
         "Exit: Ctrl+D or EOF.\n\n");
  numLines = numWords = numChars = 0;
  while((c=getchar()) != EOF){
    ++numChars;
    if(c == '\n') ++numLines;
    if(c==' ' || c=='\n' || c=='\t') insideWord = false;
    else if(!insideWord){
      insideWord = true;
      ++numWords;
    }
  }
  printf("Number of lines = %d, words = %d, characters = %d\n",
    numLines, numWords, numChars);
  return 0;
}
```

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 ! sizeof	right to left
a*b a/b a%b a+b a−b < <= > >= == != && ||	left to right
= += −= *= /= %=	right to left
,	left to right

Ordinals

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 brances, unnecessary divisions

Ordinals

ordinal2.c

```c
#include <stdio.h>
int main(void) {
  printf("Number: ");
  int number;
  scanf("%d", &number);
  if(number == 0) printf("0");
  else {
    printf("%d", number);
    if(number>10 && number<21) printf("th");
    else switch(number%10) {
      case 1: printf("st"); break;
      case 2: printf("nd"); break;
      case 3: printf("rd"); break;
      default: printf("th");
    }
  }
  return 0;
}
```

Ordinals

- switch(*expression*) *statement*
- *expression* is int type
- *statement* can contain
  - multiple case *constant-expression: statement*s,
  - zero or more default: *statement*s
- 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)

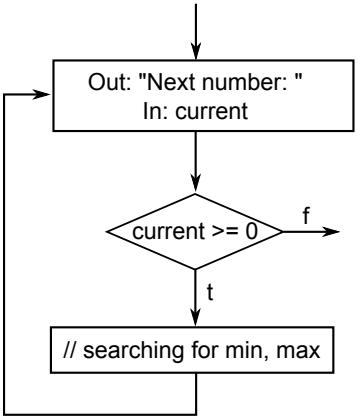
```c
#include <stdio.h>

int main(void) {
   printf("Type non-negative integers and\n"
          "the program finds their minimum and maximum.\n"
          "Exit by entering a negative number.\n");
   int quantity=0, current; // current does not need initialization
   int min, max;
   printf("Next number: "); // first occurrence of the code snippet
   scanf("%d", &current);
   while(current >= 0) {
      if(quantity == 0) min = max = current; // a selection is missing from here
      else if(current > max) max = current;
      else if(current < min) min = current;
      quantity++;
      printf("Next number: "); // second occurrence of the code snippet
      scanf("%d", &current);
   }
   if(quantity > 0) printf("Minimum: %d\nMaximum: %d\n", min, max);
   else printf("You did not enter any numbers.\n");
   return 0;
}
```

Minimum and maximum search

minmax3.c

```c
#include <stdio.h>
int main(void) {
  printf("Type non-negative integers and\n"
         "the program finds their minimum and maximum.\n"
         "Exit by entering a negative number.\n");
  int quantity=0, current;
  int min, max;
  // operator ,
  while(printf("Next number: "), scanf("%d", &current), current>=0) {
    if(!quantity) min = max = current; // operator !
    else if(current > max) max = current;
    else if(current < min) min = current;
    quantity++;
  }
  // logical expression
  if(quantity) printf("Minimum: %d\nMaximum: %d\n", min, max);
  else printf("You did not enter any numbers.\n");
  return 0;
}
```

Minimum and maximum search



Operator ,

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

Minimum and maximum search

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) ...`