

# Programming basics

## (GKNB\_INTA023)

Hatwagner F. Miklós, PhD.

Széchenyi István University, Győr, Hungary

September 12, 2020

# Square numbers

Task: print the squares of the first 10 natural numbers!

squares1.c

```
#include <stdio.h>
int main(void) {
    printf("Squares of natural numbers\n\n");
    printf("1^2=1\n");
    printf("2^2=4\n");
    printf("3^2=9\n");
    printf("4^2=16\n");
    printf("5^2=25\n");
    printf("6^2=36\n");
    printf("7^2=49\n");
    printf("8^2=64\n");
    printf("9^2=81\n");
    printf("10^2=100\n");
    return 0;
}
```

# Square numbers

## Output

Squares of natural numbers

$$1^2=1$$

$$2^2=4$$

$$3^2=9$$

$$4^2=16$$

$$5^2=25$$

$$6^2=36$$

$$7^2=49$$

$$8^2=64$$

$$9^2=81$$

$$10^2=100$$

Problems:

- We have calculated the results, not the computer!
- Too much repeating, similar lines of code
- Very easy to make mistakes, but hard to realize them

# Square numbers

Task: do the computer calculate the square numbers!

squares2.c

```
1  #include <stdio.h>
2  int main(void) {
3      printf("Squares of natural numbers\n\n");
4      printf("%d^2=%d\n", 1, 1*1);
5      printf("%d^2=%d\n", 2, 2*2);
6      printf("%d^2=%d\n", 3, 3*3);
7      printf("%d^2=%d\n", 4, 4*4);
8      printf("%d^2=%d\n", 5, 5*5);
9      printf("%d^2=%d\n", 6, 6*6);

14     printf("%d^2=%d\n", 10, 10*10);
15     return 0;
16 }
```

# Square numbers

## Output

Squares of natural numbers

```
1^2=1
2^2=4
3^2=9
4^2=16
5^2=25
6^2=36
7^2=49
8^2=64
9^2=81
10^2=100
```

Literals: constant values typed in the source code

- Integer constants
- Character constants: between `'`-s
- String constants: between `"`-s

Operator

Expression

String literal

Integer literals

The diagram shows the code `printf("%d^2=%d\n", 5, 5*5);` with several annotations. A red bracket under the first argument `"%d^2=%d\n"` is labeled "String literal". A blue arrow points to the `*` operator in `5*5` and is labeled "Operator". A brown arrow points to the entire `5*5` expression and is labeled "Expression". Three green arrows point to the `5` characters in `5*5` and are collectively labeled "Integer literals".

# Square numbers

Expression: produces a value by using literals (constants), variables and operators

format string      The further arguments  
(parameters) must correspond with  
the conversion specifiers

```
printf("%d^2=%d\n", 5, 5*5);
```

conversion    Escape  
specification sequence

Some arithmetic operators

Operator	Description	Example
+	Addition	$5 + 3 == 8$
-	Subtraction	$5 - 3 == 2$
*	Multiplication	$5 * 3 == 15$
/	Integer division ( $\rightarrow$ quotient)	$5/3 == 1$
%	Modulo ( $\rightarrow$ remainder)	$5\%3 == 2$

# Square numbers

## squares3.c

```
1 #include <stdio.h>
2 int main(void) {
3     int base;
4     printf("Squares of natural numbers\n\n");
5     base = 1;
6     printf("%d^2=%d\n", base, base*base);
7     base = base + 1;
8     printf("%d^2=%d\n", base, base*base);
9     base = base + 1;
10    printf("%d^2=%d\n", base, base*base);
11    base = base + 1;

24    printf("%d^2=%d\n", base, base*base);
25    return 0;
26 }
```

## Variables

- Eg. `int base;`
- type
  - the nature of data (numeric, text)
  - data representation
  - possible operations
- memory area
  - stores the value according to the type
  - the initial value of local variables (in blocks between `{` and `}`) is undefined, “garbage”
- name, id (should refer to its goal)



# Square numbers

## Naming rules

- First character: lower-, uppercase letter or \_
- Further characters: the same set of characters and digits as well
- Cannot be a reserved keyword or identifier
- Case sensitive
- Recommendation: do not start the names with one or two \_ characters
- Number of significant characters

### What's the problem?

John Doe  
12\_ Monkeys  
Cool!  
auto

### OK

john\_doe  
John\_Doe  
johnDoe

# Square numbers

The most important integer types (fixed point arithmetic)

Type	Description
char	Generally signed, 8 bit integer
signed char	Signed 8 bit integer
unsigned char	Unsigned (non negative) 8 bit integer
short	
signed short	Signed short integer
signed short int	
unsigned short	Unsigned short integer
unsigned short int	
signed	
int	Signed integer
signed int	
unsigned	Unsigned integer
unsigned int	
long	
signed long	Signed long integer
signed long int	
unsigned long	Unsigned long integer
unsigned long int	

## Remarks:

- (Type) modifiers: `signed/unsigned`, `short/long`
- Storage of integer literals: `int`
- Size of type `char` is always 1 byte, but character literals are stored as `int`!
- Sign of `char` depends on the platform / compiler, but it is usually signed (can be modified)
- `1 == sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)`, where `sizeof` is an operator that specifies the size of a type / variable in bytes

# Square numbers

## Variable definition

- General usage: *type variable\_list*;
- Defines a name, type and allocates memory to store the data
- Eg. `int x; int i, j, k; unsigned int y;`

## Assignment

- Operator: `=`
- *lvalue = rvalue*;
- changes the value of *lvalue* to *rvalue*
- *lvalue*: generally a variable (it cannot be a literal or an array, but a specific element of an array is allowed)

Result: the rows of our program are very similar, can be copied

Problem: many repeating code parts → let the computer repeat the same parts!

# Square numbers

squares4.c

```
#include <stdio.h>
int main(void) {
    int base;
    printf("Squares of natural numbers\n\n");
    base = 1;
    while(base <= 10) {
        printf("%d^2=%d\n", base, base*base);
        base = base + 1;
    }
    return 0;
}
```

## Relational operators

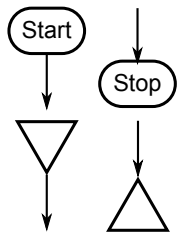
Operator	Description
==	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

while loop (tests the condition *before* executing the loop body)

```
while(condition) {  
    activities  
}
```

The *body* (repeated part) of the loop may contain

- a single statement
- a group of statements (block between { and })



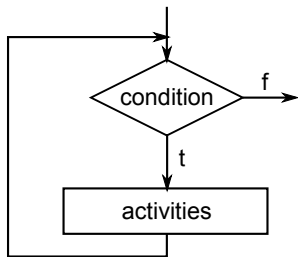
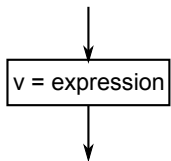
## Signals of start and end

- Triangle version: description of algorithm parts
- A program must contain exactly one start and one end point
- The start point must have a sole outgoing arrow only. The end point. . .

## Signal of direction

- May diverge only at conditions
- Shows the execution order of instructions



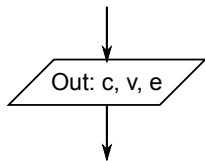


## Signal of assignment

- One arrow comes in, one goes out
- The value and type of the *expression* must be defined, then this value is going to be assigned to variable *v*

• Multiple assignments can be grouped together  
Loop (tests the condition before executing the loop body)

- Loop body: the repeated activities
- The loop body must affect the condition → infinite loop?
- The loop body may never be executed



Signal of output activity (print)

- One arrow comes in, one goes out
- The values of the  $c$  constant,  $v$  variable and  $e$  expression must be printed

# Square numbers

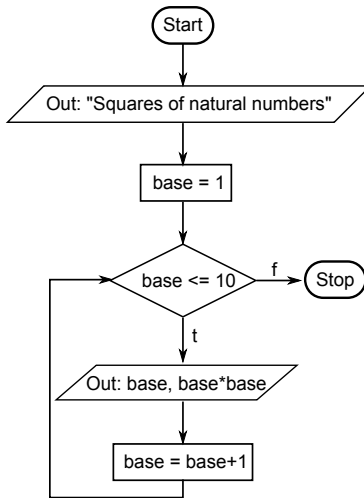


Table of data structure: the flow chart defines neither the type nor the goal of variables

Goal	Name	Type	Nature
Base of power	base	integer	work/output

# Even, odd

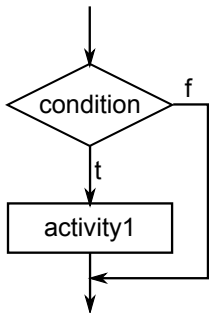
Make a decision: is a specific number odd or even?

even1.c

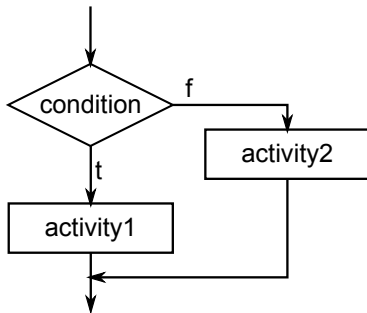
```
#include <stdio.h>

int main(void) {
    int number;
    number = 42;
    if (number % 2 == 0) {
        printf("The number is even.\n");
    } else {
        printf("The number is odd.\n");
    }
    return 0;
}
```

Conditions



*if(condition) activity1*



*if(condition) activity1  
else activity2*

Nested if-else statements are possible for multiple outcomes

What is a *pointer*, and what is it good for?

- A type suitable to store a memory address
- Several subtypes exist to express the type of the stored data
- In the technical sense, they are non-negative integer numbers
- Pointer definition: *base\_type\* name*;

## Example

```
int i;           // integer variable

int* pi;         // pointer variable that stores
                 // the address of an integer variable
```

# Pointers

The *address* of a variable can be obtained by operator **&**

The *content* of a variable *at a specific address* can be obtained by operator **\*** (indirection)

## pointer.c

```
#include <stdio.h>
int main(void) {
    int i;
    int* pi;
    i = 42;
    printf("The value of 'i' is: %d\n", i);
    printf("The address of 'i' is: %p\n", &i);
    pi = &i;
    printf("The value at address 'pi' is: %d\n", *pi);
    printf("The value of 'pi' is: %p\n", pi);
    printf("The address of 'pi' is: %p\n", &pi);
    return 0;
}
```



## Output

The value of 'i' is: 42

The address of 'i' is: 0x7ffe2c9fe634

The value at address 'pi' is: 42

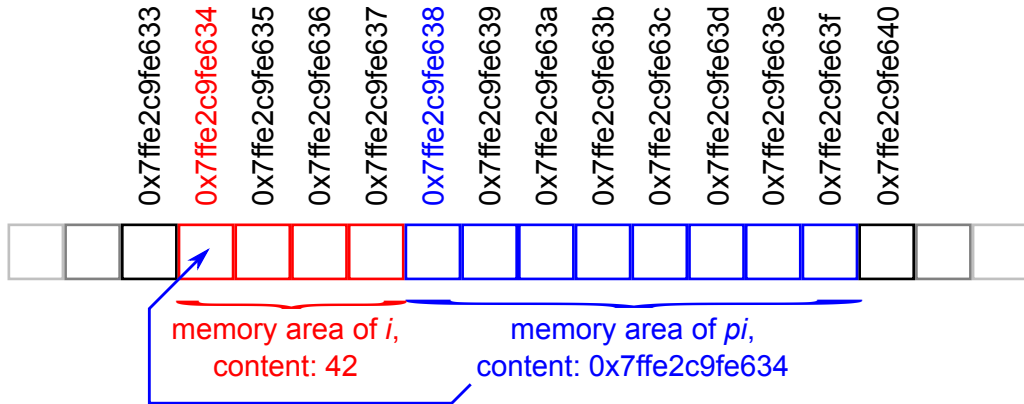
The value of 'pi' is: 0x7ffe2c9fe634

The address of 'pi' is: 0x7ffe2c9fe638

The format specifier of a pointer is: `%p`

```
printf("The address of 'i' is: %p\n", &i);
```

# Pointers



# Even, odd

Task: let the program read the number to analyze!

even2.c

```
#include <stdio.h>

int main(void) {
    int number;
    printf("Type an integer and the program decides "
           "is it even or odd.\n\n");
    scanf("%d", &number);
    if(number%2 == 0) {
        printf("The number is even.\n");
    } else {
        printf("The number is odd.\n");
    }
    return 0;
}
```

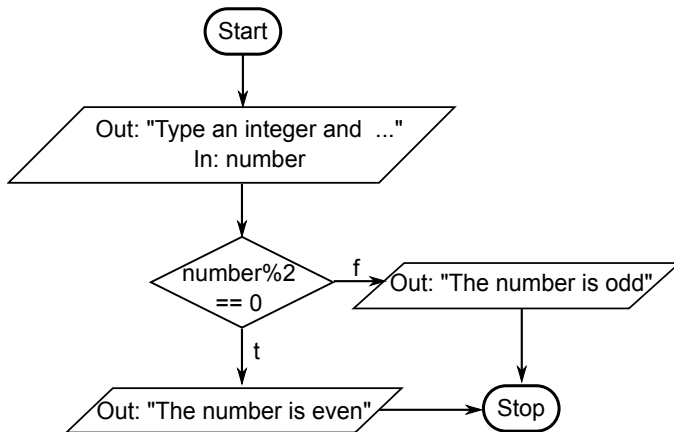
```
printf("Type an integer and the program decides "  
      "is it even or odd.\n\n");
```

The preprocessor merges the string literals if they are separated by only a *whitespace* character.

```
scanf("%d", &number);
```

Reading from the standard input

- **scanf** is the “inverse” of `printf`
- Its first argument is the format string (similarly to `printf`)
- The following arguments must match the applied format specifiers
- The *address* of variables that store the read, converted values must be given



## Order of instructions (expressions)

- parenthesis
- precedence

Operator	Associativity
sizeof	right to left
* / %	left to right
+ -	left to right
< <= > >=	left to right
== !=	left to right
=	right to left

## Control structures

- Sequence (Statements are executed in a specified order. No statement is skipped and no statement is executed more than once.)
- Repetition (loop, iteration)
- Selection (condition)