C programming

- Introduction
- The basics of algorithms
- Structure of a C code, compilation step
- Constant, variable type, variable scope
- Expression and operators: assignment, arithmetic operators, comparison, type conversion
- Control statement: conditional and loops
- Tabular data
- Functions
- Structures, strings and pointers
- Input/Output
Sources:

• **On the Web:**

• Slides will be available on:
Computers and programs

• A computer is a processing unit:
  – performs a suite of operations
  – stores the results in its memory

• The suite of operation is user defined and is called a program
  – it is the duty of the user to design the proper algorithm which must be efficient and “possibly” error free

• A computer does not think for you!
Programming language

• Two main types of languages:
  – iterative (C, Pascal, Fortran, PERL...)
  – object oriented (C++, ADA, COBOL, Java...)

• We will use C in this lecture
  – standard and widely used (Linux/Unix are written in C for example)
  – well adapted to scientific computing
  – structure of a C code follows the standard problem solving logic
Algorithm

• Definition: an algorithm is a finite sequence of instructions, a logic and explicit step-by-step procedure for solving a problem starting from a known beginning.
  – the number of instructions must be finite
  – to write the algorithm you first must know how to solve the problem!
  – the solution must follow a logical path: the order of the instructions IS important
Here is a real life example of an algorithm.

The only missing part is the known beginning: “in case of fire”
A simple algorithm

Algorithm Sum the integers from 1 to N

\[\begin{align*}
  sum & \leftarrow 0; \\
  x & \leftarrow 1; \\
  \text{while } x \leq N \text{ do} \\
  & \quad sum \leftarrow sum + x; \\
  & \quad x \leftarrow x + 1; \\
  \text{end while} \\
  \text{print } sum;
\end{align*}\]

- Defines the logical order of the operation
- Identifies the variable that are needed to solve the problem
- Free from programming language syntax = general, can be transposed in any language that follows the iterative scheme
Algorithm and flowchart

A flowchart is a graphical representation of an algorithm. Usually, we start with drawing the flowchart and then proceed to the algorithm.
Algorithm and flowchart

This flowchart handles a line of characters and counts the numbers (numeric characters) in the line:

- start from the beginning of the line
- loop over all characters in the line
  - tests if a number
  - update counter if yes
  - tests eol character
Algorithm and flowchart

This is the algorithm associated to the previous flowchart

```
Algorithm Count the number of numeric character in a line

count ← 0;
i ← 0;
ch ← line[i];
while ch ≠ \n do
    if ch in [0,9] then count ← count + 1;
i ← i + 1;
    ch ← line[i];
end while
print count;
```
Algorithm and flowchart

1. Start
2. Read number
3. Set largest = number
4. Repeat:
   a. Ask: Any more numbers?
   b. If yes, read number
   c. If no, write largest and stop
5. Else:
   a. Ask: Is number > largest?
   b. If yes, set largest = number
   c. Go back to step 4
Algorithm Largest number

read number
largest ← number;
while new number do
    read number
    if number > largest then largest ← number;
end while
print largest;
load library, define constants...

declaring a function:
how is it used?

main program and its
instructions

defining a function:
what is it doing?
Structure of a C code (2)

• It is important to keep a proper organization of your code: as long as you are not experienced keep the scheme presented before.

• Keep a good logic: a computer does not think for you, it just computes whatever you ask him to compute!

• ALWAYS declare what you are using (functions, variables etc.)!
A simple C code

- first step: load a library that allows to interact with the screen or keyboard
- second step: start the main program block
- third step: use the `printf` function (from the `stdio.h` library) to print Hello
- fourth step: exit the program with a given error code
- last step: close the main program block
Another example (a bit more complicated)
reading from and writing to

• C provides 2 methods that enable you to read from the keyboard and write to the standard output (screen). Both are part of stdio:
  – scanf(“%i”,&x) to read from the keyboard
  – printf(“%i”,x) to write to the screen
• %i for integers, %f for floats
• Functions return 1 is everything is OK
Going from C code to the actual program (under Unix/Linux)

1) A C program needs to be compiled (using gcc for ex.) before it can be run on the computer

2) 

3)
Variables in C

• Variables contain values that must be kept during the completion of a program (storage) for future use.
• In C, a variable MUST be declared before it can be used.
• Variables can be declared at the start of any block code, but most are found at the start of each function (main inclusive)
• Local variables are created when the function is called and destroyed on return from that function (see scope and range later)
How to declare a variable?

• a declaration begins with the type, followed by the name of the variable:

```c
int x; //declares an integer variable x
```

• More than one variable can be assigned and a starting value can be given:

```c
float x,y=-2.5,tab[20];
```
Types in C (32 bits machine):

• int = integer value 4 bits
• float = real value 4 bits
• char = single character
• short = integer value 2 bits
• long int = integer value 4 bits
• unsigned = only positive values
• double = high precision real value 8 bits
# Types in C

<table>
<thead>
<tr>
<th>C type</th>
<th>Size (bytes)</th>
<th>Lower bound</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>short int</td>
<td>2</td>
<td>−32768</td>
<td>+32767</td>
</tr>
<tr>
<td>unsigned short int</td>
<td>2</td>
<td>0</td>
<td>65536</td>
</tr>
<tr>
<td>(long) int</td>
<td>4</td>
<td>−2^31</td>
<td>+2^31 − 1</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>−3.2 × 10^{±38}</td>
<td>+3.2 × 10^{±38}</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>−1.7 × 10^{±308}</td>
<td>+1.7 × 10^{±308}</td>
</tr>
</tbody>
</table>
Variable names

• Each variable has a value and a name.
• There is a limitation on what the name of a variable can be: it must start with a letter or underscore (counter, _counter)
• no space or special character
• length < 8 character
• Also, it is forbidden to use one of C’s keyword as a variable name as main, switch, while...but this is common sense
C language keywords

<table>
<thead>
<tr>
<th>auto</th>
<th>else</th>
<th>register</th>
<th>union</th>
</tr>
</thead>
<tbody>
<tr>
<td>break</td>
<td>enum</td>
<td>return</td>
<td>unsigned</td>
</tr>
<tr>
<td>case</td>
<td>extern</td>
<td>short</td>
<td>void</td>
</tr>
<tr>
<td>char</td>
<td>float</td>
<td>signed</td>
<td>volatile</td>
</tr>
<tr>
<td>const</td>
<td>for</td>
<td>size of</td>
<td>while</td>
</tr>
<tr>
<td>continue</td>
<td>goto</td>
<td>static</td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>if</td>
<td>struct</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td>int</td>
<td>switch</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>long</td>
<td>typedef</td>
<td></td>
</tr>
<tr>
<td>restrict</td>
<td>_Bool</td>
<td>_Complex</td>
<td>_Imaginary</td>
</tr>
</tbody>
</table>
# Special Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Name</th>
<th>Character</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>,</td>
<td>Comma</td>
<td>&amp;</td>
<td>Ampersand</td>
</tr>
<tr>
<td>.</td>
<td>Period</td>
<td>^</td>
<td>Caret</td>
</tr>
<tr>
<td>;</td>
<td>Semicolon</td>
<td>*</td>
<td>Asterisk</td>
</tr>
<tr>
<td>:</td>
<td>Colon</td>
<td>-</td>
<td>Minus Sign</td>
</tr>
<tr>
<td>?</td>
<td>Question Mark</td>
<td>+</td>
<td>Plus Sign</td>
</tr>
<tr>
<td>'</td>
<td>Apostrophe</td>
<td>&lt;</td>
<td>Opening Angle (Less than sign)</td>
</tr>
<tr>
<td>“</td>
<td>Quotation Marks</td>
<td>&gt;</td>
<td>Closing Angle (Greater than sign)</td>
</tr>
<tr>
<td>!</td>
<td>Exclamation Mark</td>
<td>(</td>
<td>Left Parenthesis</td>
</tr>
<tr>
<td></td>
<td>Vertical Bar</td>
<td>)</td>
<td>Right Parenthesis</td>
</tr>
<tr>
<td>/</td>
<td>Slash</td>
<td>[</td>
<td>Left Bracket</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
<td>]</td>
<td>Right Bracket</td>
</tr>
<tr>
<td>~</td>
<td>Tiilde</td>
<td>{</td>
<td>Left Brace</td>
</tr>
<tr>
<td>-</td>
<td>Underscore</td>
<td>}</td>
<td>Right Bracket</td>
</tr>
<tr>
<td>$</td>
<td>Dollar Sign</td>
<td>#</td>
<td>Number Sign</td>
</tr>
<tr>
<td>%</td>
<td>Percentage Sign</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td>Meaning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>'a'</code></td>
<td>Audible Alert (Bell)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>'b'</code></td>
<td>Backspace</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>'f'</code></td>
<td>Formfeed</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>'n'</code></td>
<td>New Line</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>'r'</code></td>
<td>Carriage Return</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>'t'</code></td>
<td>Horizontal tab</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>'v'</code></td>
<td>Vertical Tab</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>'\'</code></td>
<td>Single Quote</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>''</code></td>
<td>Double Quote</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>''</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>'?'</code></td>
<td>Question Mark</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code> '\'</code></td>
<td>Back Slash</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>'0'</code></td>
<td>Null</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Local and global variables

- Local variables are declared within the body of a function and can only be used within that same function.
- Usually, for an external variable to be known for a function, it must be passed as argument
- A variable can also be declared globally and so it is available to all functions
#include <stdio.h>

/* Here I define a global variable */
int counter1 = 0;

//function that will be used later
int adding(int i, int j);

int main()
{
    //Here I define also a global variable
    int counter2 = 1;
    int i;

    for (i = 1; i <= 10; i++)
    {
        //Here j is a local variable known only inside the for block
        int j;
        counter1++;
    }
    i = adding(counter1, counter2);

    return 0;
}

int adding(int i, int j)
{
    //k is a local variable know only inside the function
    int k;
    k = i + j;
    return k;
}
Another example
Static variables

A static can only be accessed from the function in which it was declared, like a local variable. The static variable is not destroyed on exit from the function, instead its value is preserved, and becomes available again when the function is next called.

Static variables are declared as local variables, but the declaration is preceded by the word static.
Example

static int counter;

• Static variables can be initialized as normal, the initialization is performed once only, when the program starts up.
External variable

• Where a global variable is declared in one file, but used by functions from another, then the variable is called an external variable in these functions, and must be declared as such.

• The declaration must be preceded by the word extern. The declaration is required so the compiler can find the type of the variable without having to search through several source files for the declaration.
Constants

A constant value is the one which does not change during the execution of a program. C supports several types of constants.

1. Integer Constants
2. Real Constants
3. Single Character Constants
4. String Constants
Constants

• The const keyword is to declare a constant, as shown below:
  
  \[
  \text{int const } a = 1; \\
  \text{const int } a = 2;
  \]

Note:

• You can declare the const before or after the type. Choose one and stick to it.

• It is usual to initialize a const with a value as it cannot get a value any other way.
Working with variables

• Variables are useful to store information
• The next step is to use this information in order to compute a required quantity, compare values or manipulate a string or a file
• There are many operators in C to permit it
• However the user must be careful!
  – operators have precedence
  – some work from left to right, some from right to left
<table>
<thead>
<tr>
<th>Description</th>
<th>Represented By</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Parenthesis</td>
<td>⟨⟩ □</td>
</tr>
<tr>
<td>1 Structure Access</td>
<td>. − ⟶</td>
</tr>
<tr>
<td>2 Unary</td>
<td>! − ++ −− − ∗ &amp;</td>
</tr>
<tr>
<td>3 Multiply, Divide, Modulus</td>
<td>∗ / %</td>
</tr>
<tr>
<td>4 Add, Subtract</td>
<td>+ −</td>
</tr>
<tr>
<td>5 Shift Right, Left</td>
<td>&gt;&gt; &lt;&lt;</td>
</tr>
<tr>
<td>6 Greater, Less Than, etc</td>
<td>&gt; &lt; =</td>
</tr>
<tr>
<td>7 Equal, Not Equal</td>
<td>== !=</td>
</tr>
<tr>
<td>8 Bitwise AND</td>
<td>∧</td>
</tr>
<tr>
<td>9 Bitwise Exclusive OR</td>
<td>‖</td>
</tr>
<tr>
<td>10 Bitwise OR</td>
<td></td>
</tr>
<tr>
<td>11 Logical AND</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>12 Logical OR</td>
<td></td>
</tr>
<tr>
<td>13 Conditional Expression</td>
<td>?:</td>
</tr>
<tr>
<td>14 Assignment</td>
<td>= += −= etc</td>
</tr>
<tr>
<td>15 Comma</td>
<td>,</td>
</tr>
</tbody>
</table>
## Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition or Unary Plus</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction or Unary Minus</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>%</td>
<td>Modulus Operator</td>
</tr>
</tbody>
</table>

Examples of arithmetic operators are:

- \( x + y \)
- \( x - y \)
- \( -x + y \)
- \( a + b + c \)
- \( -a + b \)
2. Relational Operators

Often it is required to compare the relationship between operands and bring out a decision and program accordingly. This is when the relational operator come into picture. C supports the following relational operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>is less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>is less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>is greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>is greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>is equal to</td>
</tr>
<tr>
<td>!=</td>
<td>is not equal to</td>
</tr>
</tbody>
</table>

It is required to compare the marks of 2 students, salary of 2 persons, we can compare them using relational operators.

A simple relational expression contains only one relational operator and takes the following form.

```
exp1 relational operator exp2
```

Where `exp1` and `exp2` are expressions, which may be simple constants, variables or combination of them. Given below is a list of examples of relational expressions and evaluated values.

- `6.5 <= 25` TRUE
- `-65 > 0` FALSE
- `10 < 7 + 5` TRUE

Relational expressions are used in decision making statements of C language such as if, while and for statements to decide the course of action of a running program.
Relational instructions and logical representation

- the relation operation returns a TRUE-FALSE answer
- This answer is translated into a numeric value
- 0 represents FALSE
- 1 or any value besides 0 represents TRUE
3. Logical Operators

C has the following logical operators, they compare or evaluate logical and relational expressions.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>Logical AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Logical NOT</td>
</tr>
</tbody>
</table>

**Logical AND (&&)**

This operator is used to evaluate 2 conditions or expressions with relational operators simultaneously. If both the expressions to the left and to the right of the logical operator is true then the whole compound expression is true.

**Example**

```c
a > b && x = 10
```

The expression to the left is `a > b` and that on the right is `x = 10` the whole expression is true only if both expressions are true i.e., if `a` is greater than `b` and `x` is equal to 10.
**Logical OR (||)**

The logical OR is used to combine 2 expressions or the condition evaluates to true if any one of the 2 expressions is true.

**Example**

```
a < m || a < n
```

The expression evaluates to true if any one of them is true or if both of them are true. It evaluates to true if a is less than either m or n and when a is less than both m and n.

**Logical NOT (!)**

The logical not operator takes single expression and evaluates to true if the expression is false and evaluates to false if the expression is true. In other words it just reverses the value of the expression.

**For example**

```
! (x >= y)  the NOT expression evaluates to true only if the value of x is neither greater than or equal to y
```
4. Assignment Operators

The Assignment Operator evaluates an expression on the right of the expression and substitutes it to the value or variable on the left of the expression.

Example

\[ x = a + b \]

Here the value of \( a + b \) is evaluated and substituted to the variable \( x \).

In addition, C has a set of shorthand assignment operators of the form.

\[ \text{var oper = exp; } \]

Here \text{var} is a variable, \text{exp} is an expression and \text{oper} is a C binary arithmetic operator. The operator \text{oper =} is known as shorthand assignment operator.
**Example**

\[ x = x + 1 \] is same as \[ x := x + 1 \]

The commonly used shorthand assignment operators are as follows:

**Shorthand assignment operators**

<table>
<thead>
<tr>
<th>Statement with simple assignment operator</th>
<th>Statement with shorthand operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ a = a + 1 ]</td>
<td>[ a += 1 ]</td>
</tr>
<tr>
<td>[ a = a - 1 ]</td>
<td>[ a -= 1 ]</td>
</tr>
<tr>
<td>[ a = a \times (n+1) ]</td>
<td>[ a *= (n+1) ]</td>
</tr>
<tr>
<td>[ a = a / (n+1) ]</td>
<td>[ a /= (n+1) ]</td>
</tr>
<tr>
<td>[ a = a % b ]</td>
<td>[ a %= b ]</td>
</tr>
</tbody>
</table>
Type conversion: cast

• It is often useful to be able to change the type of a variable, for example from floating point to integer or changing an integer into a float
• These operations are called casts
• In C it is easily done by specifying the new type in front of the name:
  – if x is an integer, typing (float) x in the program will use x as a float
Type conversion: cast

• When using mathematical formulae, C automatically uses the type with highest precision
• But beware of the precedence of operation
• ex: if a=2.2 is a float, x=2 and y=3 are ints
  x/y*a is not the same as a*x/y
Type conversion: cast

```c
#include "stdio.h"

int main(){
    float a=2.2;
    int x=2,y=3;
    printf("case 1 : x/y*a=\%f\n and a\times/y=\%f\n",x/y*a,a*x/y);
    return 1;
}
```

```
--- example.c  All L7
Wrote /Users/sieber/t/example.c
```

gcc -o example example.c
```
\$ gcc -o example example.c
```
```
\$ ./example
```
```
case 1 : x/y*a=0.000000
 and a\times/y=1.466667
```

Handling conditions

• It is frequent that a given set of instructions (operations) must be performed only if a variable has a specific value or if a condition is fulfilled

• To handle this, C provides a few conditional operators
if statement

if (expression){
    statement;
}

...or:
if (expression){
    statement1;
} else{
    statement2;
}

...or:
if (expression){
    statement1;
} else if (expression){
    statement2;
} else{
    statement3}

expression represents a logical operation. The outcome must be 0 or 1. Ex:
    a<b
    x==1.
    etc.
if statement

```c
#include "stdio.h"

int main()
{
    int x;
    scanf("%i", &x);
    if (x%2==0){
        printf("%i is even\n", x);
    } else
        printf("%i is odd\n", x);
    return 1;
}
```
if statement

```c
#include "stdio.h"

int main(){
    int x,y;
    scanf("%i %i", &x, &y);
    if (x>y){
        printf("%i is larger than %i\n", x, y);
    } else if (x<y){
        printf("%i is lower than %i\n", x, y);
    } else {
        printf("The two numbers are equal\n");
    }
    return 1;
}
```

```
4 3
4 is larger than 3
```

```
5 7
5 is lower than 7
```

```
11
The two numbers are equal
```

if statement

```c
int main()
{
    float x,y;
    scanf("%f %f",&x,&y);

    if ((2*x-2*y) && (x>=0) && (x<=1))
        printf("the point is below the curve and in the proper interval\n");
    else if ((x<0) || (x>1))
        printf("the point is outside of the interval\n");
    else
        printf("the point is above the curve\n");

    return 1;
}
```
Switch statement

```c
int main(){
    char letter;
    int vowel=0,space=0,other=0;
    scanf("%c",&letter);

    switch(letter){
    case 'A':
        vowel++;
        break;
    case 'E':
        vowel++;
        break;
    case 'I':
        vowel++;
        break;
    case 'O':
        vowel++;
        break;
    case 'U':
        vowel++;
        break;
    case ' ': 
        space++;
        break;
    default:
        other++;
        break;
    }

    printf("%i %i %i\n",vowel,space,other);
    return 1;
}
```
Switch statement

```c
#include "stdio.h"

int main()
{
    int x, y;
    scanf("%i", &x);
    y = x % 2;
    switch (y)[
    case 0:
        printf("%i is even\n", x);
        break;
    case 1:
        printf("%i is odd\n", x);
        break;
    }
    return 1;
}
```
Switch statement

- Attention: the expression in the switch must be of integer or character type.
Ternary operator

• The \(?\) (ternary condition) operator is a more efficient form for expressing simple if statements. It has the following form:
expression1 \(?\) expression2: expression3

It simply states:
• if expression1 then expression2 else expression3
Ternary operator

```c
#include "stdio.h"

int main(){
    float x, y, z;
    scanf("%f %f", &x, &y);
    // syntax with if statement
    if (x < y){
        z = x;
    } else {
        z = y;
    }
    printf("z=%f\n", z);
    // ternary operator
    z = x < y ? x : y;
    printf("z=%f\n", z);
    return 1;
}
```

```
siebert@pluto("\{102\}"
> gcc -o example example.c
> ./example
> 3 4
> z=3.000000
> z=3.000000
> siebert@pluto("\{104\}"
```