Program: “RPNConsole”

Program created with Visual Studio 2008
to make algebraic operations with two numbers,
a calculator working in console mode using the RPN

Computer Graphics for Engineering
Luiz Fernando Martha & André Pereira

Lecture Aims:

Introduction of programming concepts for the C/C++ language by using an example of a RPN (Reversed Polish Notation) Calculator.

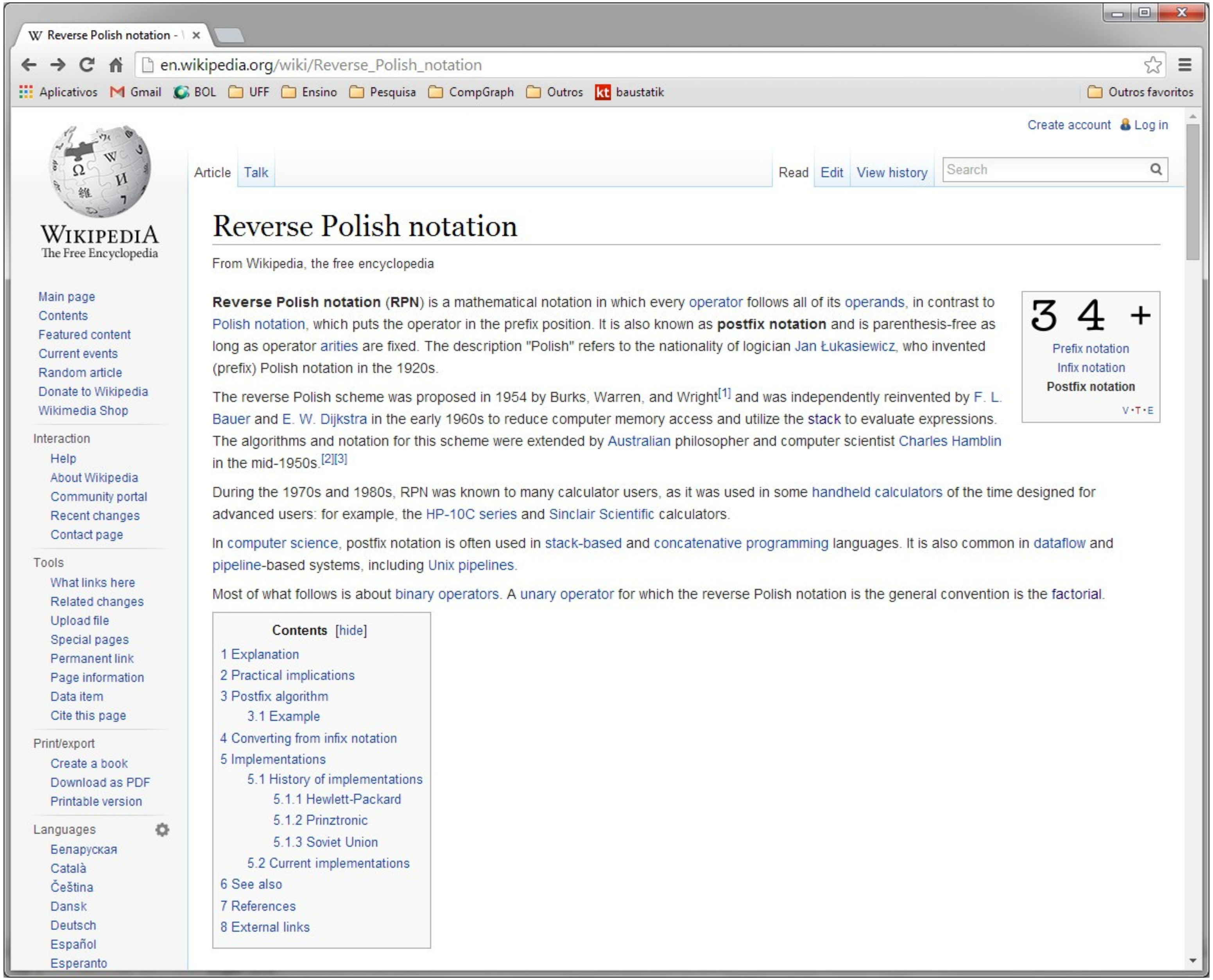
Contents/Issues Addressed:

- Introduction to C/C++ programming language.
- Introduction to the concepts of classes and objects in C++.

Capabilities/Skills:

- Know the basic resources of C/C++ language.
- Ability to create a console project in Visual Studio 2008.
- Capacity of understanding algorithms and data structures.
- Understanding the use of functions and mechanisms for passing parameters to functions in C/C++ language.
- Students should be able to develop practical applications.

What does a RPN (Reserved Polish Notation) Calculator mean?



Reverse Polish notation

From Wikipedia, the free encyclopedia

Reverse Polish notation (**RPN**) is a mathematical notation in which every operator follows all of its operands, in contrast to Polish notation, which puts the operator in the prefix position. It is also known as **postfix notation** and is parenthesis-free as long as operator arities are fixed. The description "Polish" refers to the nationality of logician Jan Łukasiewicz, who invented (prefix) Polish notation in the 1920s.

The reverse Polish scheme was proposed in 1954 by Burks, Warren, and Wright[1] and was independently reinvented by F. L. Bauer and E. W. Dijkstra in the early 1960s to reduce computer memory access and utilize the stack to evaluate expressions. The algorithms and notation for this scheme were extended by Australian philosopher and computer scientist Charles Hamblin in the mid-1950s.[2][3]

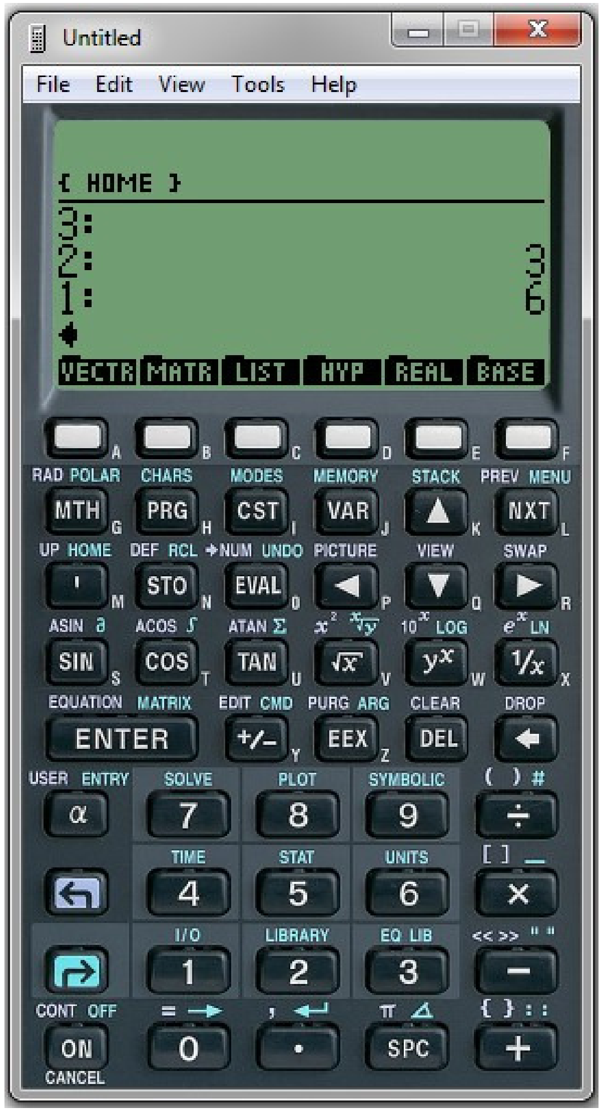
During the 1970s and 1980s, RPN was known to many calculator users, as it was used in some handheld calculators of the time designed for advanced users: for example, the HP-10C series and Sinclair Scientific calculators.

In computer science, postfix notation is often used in stack-based and concatenative programming languages. It is also common in dataflow and pipeline-based systems, including Unix pipelines.

Most of what follows is about binary operators. A unary operator for which the reverse Polish notation is the general convention is the factorial.

Contents [hide]

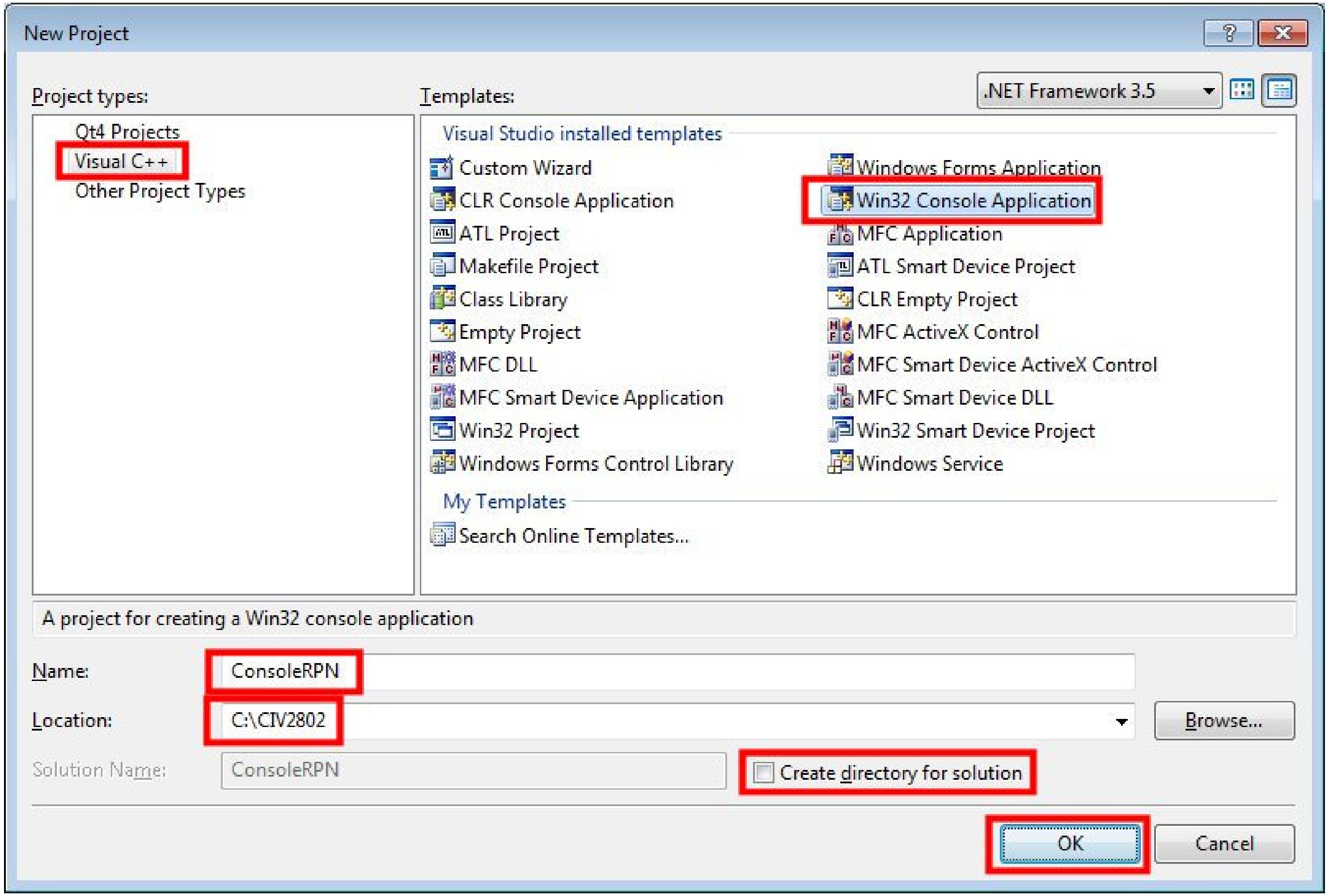
Classical Example of a calculator that utilizes the Reverse Polish Notation (RPN)


Untitled
File Edit View Tools Help
{ HOME }
3:
2: 3
1: 6
VECTR MATR LIST HYP REAL BASE
RAD POLAR CHARS MODES MEMORY STACK PREV MENU
MTH PRG CST VAR ▲ NXT
UP HOME DEF RCL →NUM UNDO PICTURE VIEW SWAP
' STO EVAL ◀ ▼ ▶
ASIN ∂ ACOS ∫ ATAN Σ x² ˣ√y 10ˣ LOG eˣ LN
SIN COS TAN √x yˣ 1/x
EQUATION MATRIX EDIT CMD PURG ARG CLEAR DROP
ENTER +/− EEX DEL ←
USER ENTRY SOLVE PLOT SYMBOLIC ( ) #
α 7 8 9 ÷
TIME STAT UNITS [ ] _
4 5 6 ×
I/O LIBRARY EQ LIB << >> " "
1 2 3 −
CONT OFF = → ; ↵ π ∡ { } : :
ON 0 • SPC +
CANCEL

How to create our own calculator using a programming language and a graphical user interface?

Creation of a new Visual Studio 2008 project type: Console

Let’s start programming a calculator without a graphical interface



New Project

Project types: Visual C++

Templates: Win32 Console Application

A project for creating a Win32 console application

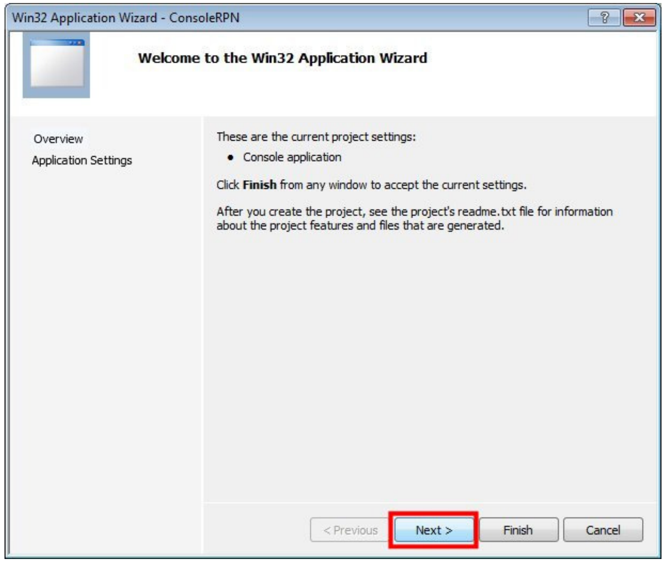
Name: ConsoleRPN

Location: C:\CIV2802

Solution Name: ConsoleRPN

□ Create directory for solution

Wizard for the creation of an “Empty” Console project type


Win32 Application Wizard - ConsoleRPN

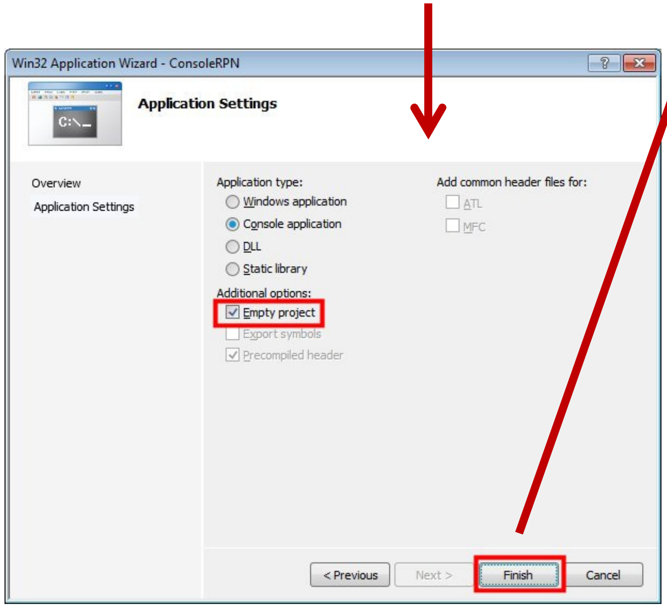
Welcome to the Win32 Application Wizard

These are the current project settings:

- Console application

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.


Win32 Application Wizard - ConsoleRPN

Application Settings

Application type:

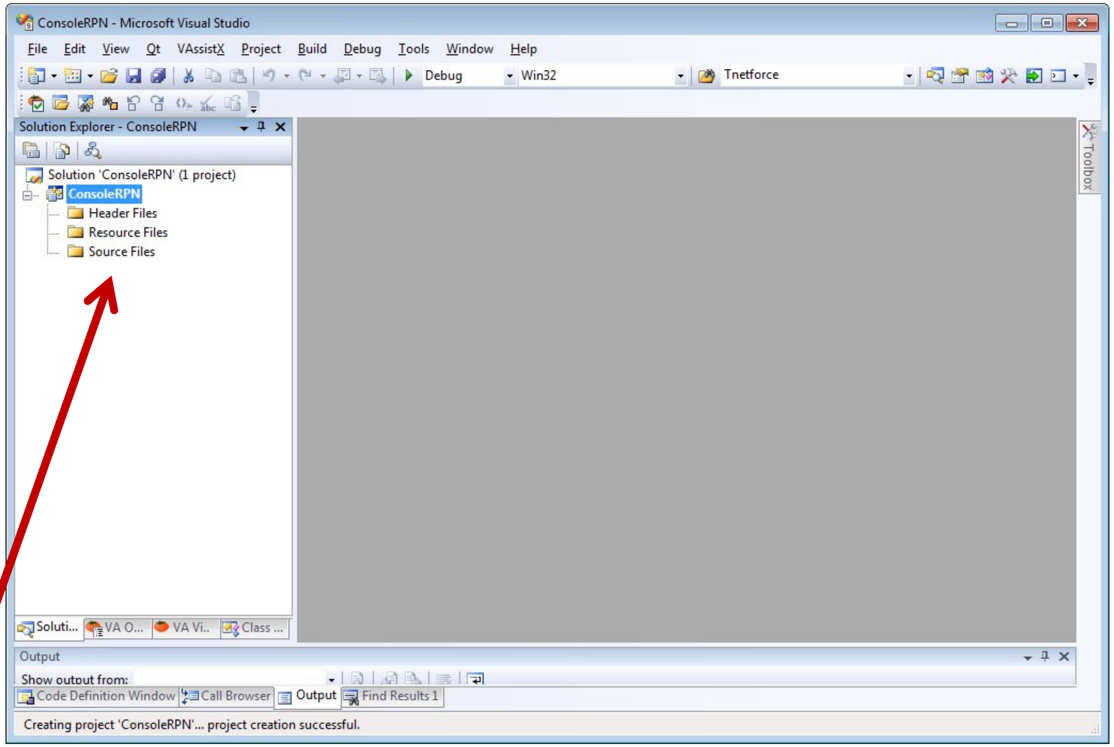
○ Windows application
● Console application
○ DLL
○ Static library

Add common header files for:

□ ATL
□ MFC

Additional options:

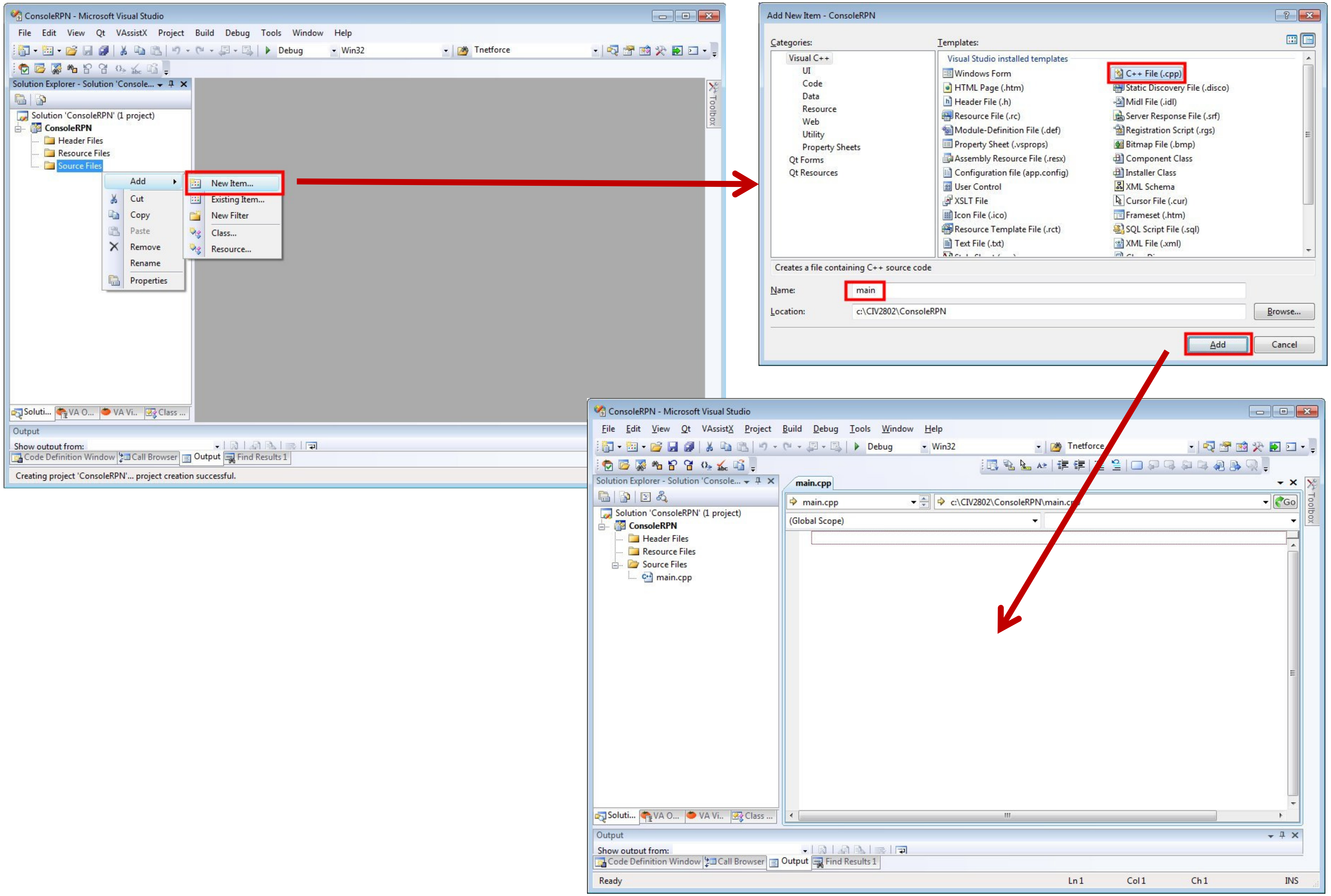
☑ Empty project
□ Export symbols
☑ Precompiled header


ConsoleRPN - Microsoft Visual Studio

Creating project 'ConsoleRPN'... project creation successful.

How to start the development of a program from scratch?

Creation of a new file, calling it by “main” and with extension “.cpp”



ConsoleRPN - Microsoft Visual Studio

Solution 'ConsoleRPN' (1 project)

Creating project 'ConsoleRPN'... project creation successful.

Add New Item - ConsoleRPN

Categories: Visual C++

Templates: Visual Studio installed templates — C++ File (.cpp)

Creates a file containing C++ source code

Name: main

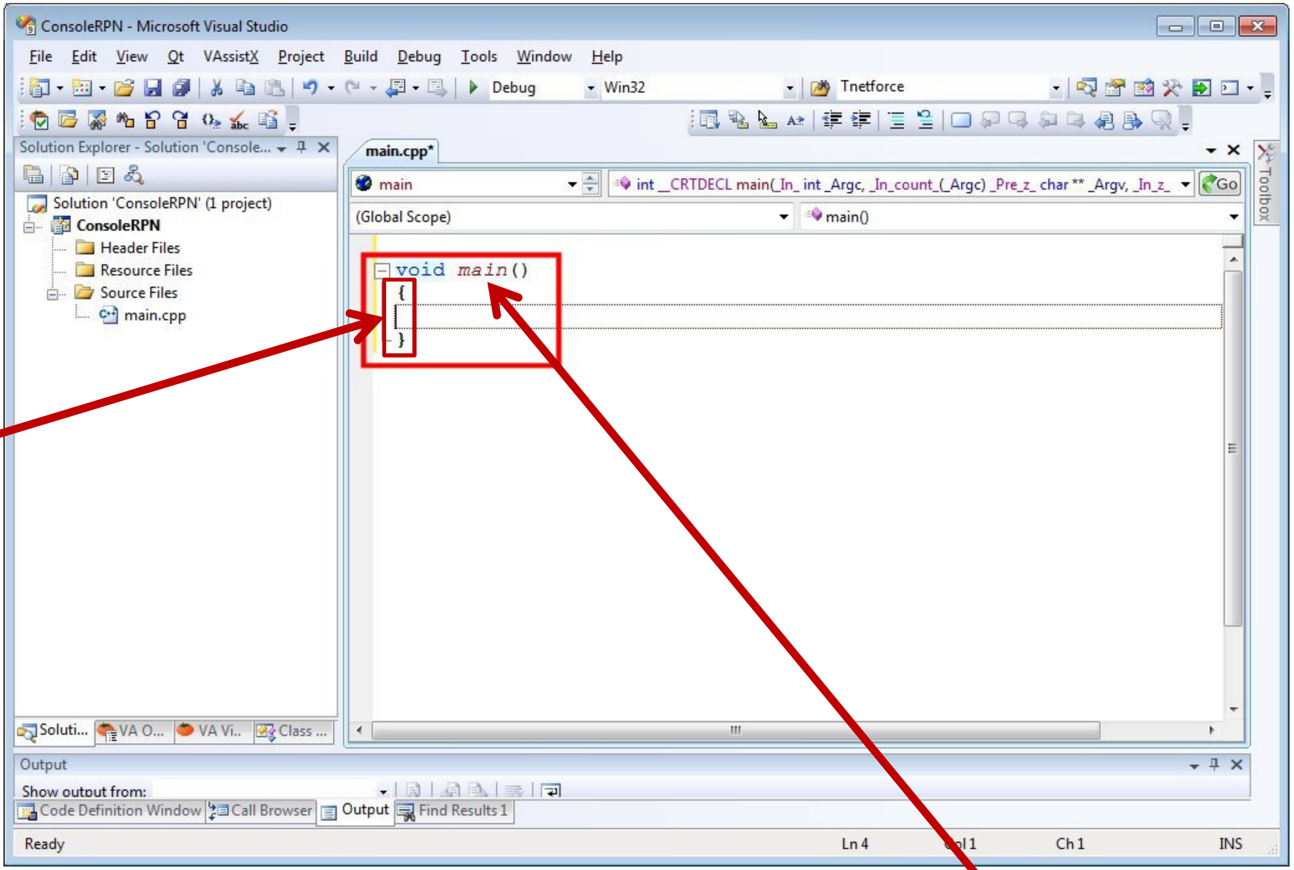
Location: c:\CIV2802\ConsoleRPN

main.cpp — c:\CIV2802\ConsoleRPN\main.cpp — (Global Scope)

Ready — Ln 1 Col 1 Ch 1 INS

Implementation of the “main.cpp” file

Basic structure of a C program


ConsoleRPN - Microsoft Visual Studio
File Edit View Qt VAssistX Project Build Debug Tools Window Help
Debug
Win32
Tnetforce
Solution Explorer - Solution 'Console...
Solution 'ConsoleRPN' (1 project)
ConsoleRPN
Header Files
Resource Files
Source Files
main.cpp
Soluti...
VA O...
VA Vi...
Class ...
main.cpp*
main
int __CRTDECL main(_In_ int _Argc, _In_count_(_Argc) _Pre_z_ char ** _Argv, _In_z_
Go
(Global Scope)
main()
Toolbox
Output
Show output from:
Code Definition Window
Call Browser
Output
Find Results 1
Ready
Ln 4
Col 1
Ch 1
INS

```
void main()
{

}
```

A pair of brackets defines a block of code

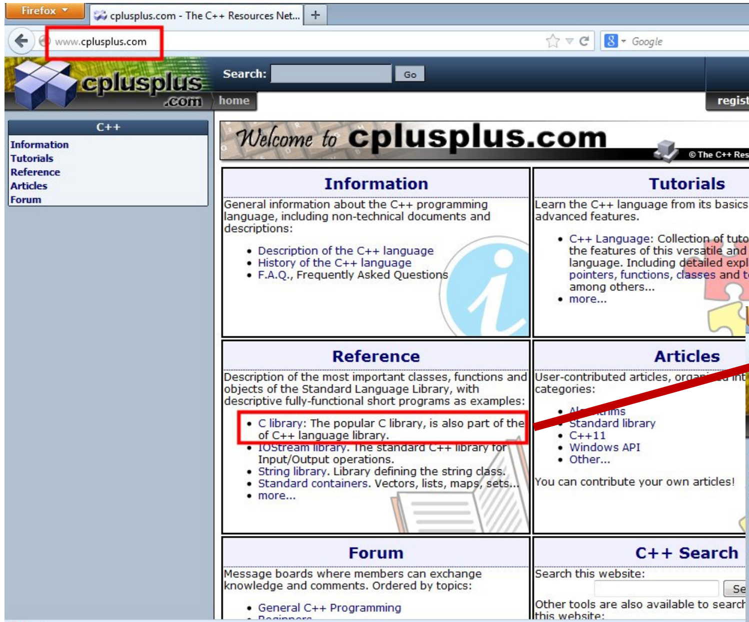
Every C program contains at least one function:
`main`
This case **`void main()`** requires no parameter and returns no parameter.

The program execution starts by calling the **`main`** function.

The main function may have different signatures

- `main()`
- `int main(int argc, char **argv)`
- `int main(int argc, char **argv, char **env)`

Documentation and reference to programming in C and C++



Information

General information about the C++ programming language, including non-technical documents and descriptions:

- Description of the C++ language
- History of the C++ language
- F.A.Q., Frequently Asked Questions

Tutorials

Learn the C++ language from its basics up to its most advanced features.

- C++ Language: Collection of tutorials covering all the features of this versatile and powerful language. Including detailed explanations of pointers, functions, classes and templates, among others...
- more...

Reference

Description of the most important classes, functions and objects of the Standard Language Library, with descriptive fully-functional short programs as examples:

- C library: The popular C library, is also part of the of C++ language library.
- IOStream library. The standard C++ library for Input/Output operations.
- String library. Library defining the string class.
- Standard containers. Vectors, lists, maps, sets...
- more...

Articles

User-contributed articles, organized int categories:

- Algorithms
- Standard library
- C++11
- Windows API
- Other...

You can contribute your own articles!

Forum

Message boards where members can exchange knowledge and comments. Ordered by topics:

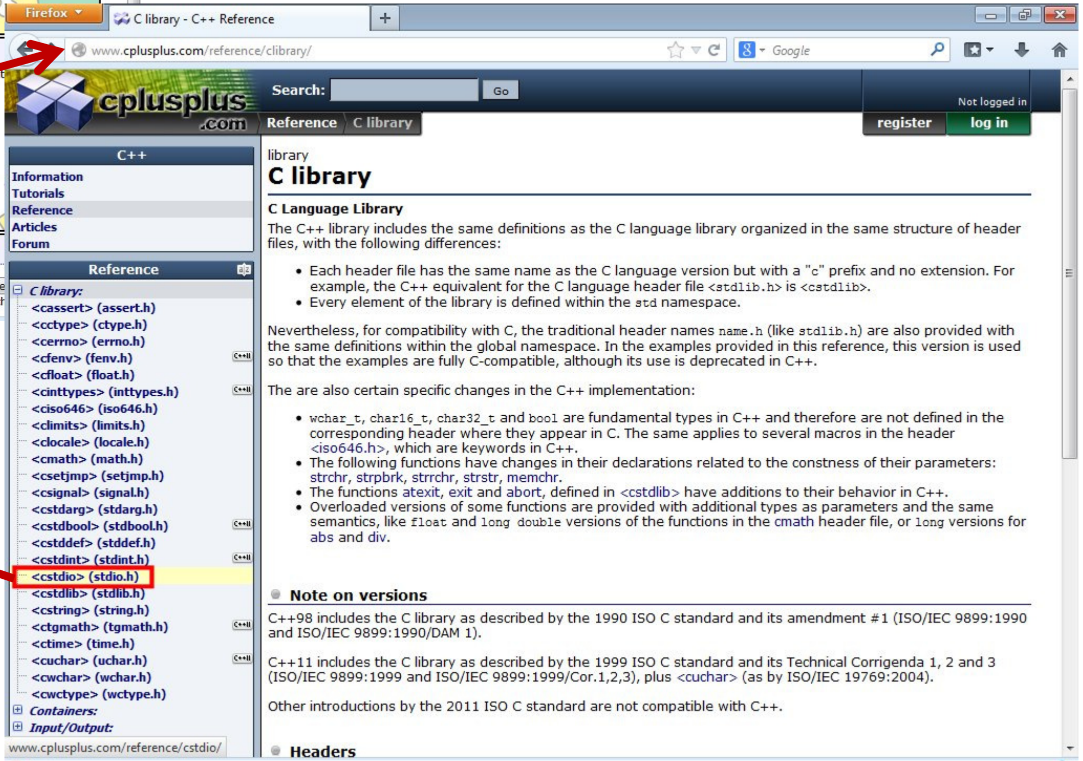
- General C++ Programming
- Beginners

C++ Search

Search this website:

Other tools are also available to search this website:

How to output data in the screen of a console program?



C library:

- <cassert> (assert.h)
- <cctype> (ctype.h)
- <cerrno> (errno.h)
- <cfenv> (fenv.h)
- <cfloat> (float.h)
- <cinttypes> (inttypes.h)
- <ciso646> (iso646.h)
- <climits> (limits.h)
- <clocale> (locale.h)
- <cmath> (math.h)
- <csetjmp> (setjmp.h)
- <csignal> (signal.h)
- <cstdarg> (stdarg.h)
- <cstdbool> (stdbool.h)
- <cstddef> (stddef.h)
- <cstdint> (stdint.h)
- <cstdio> (stdio.h)
- <cstdlib> (stdlib.h)
- <cstring> (string.h)
- <ctgmath> (tgmath.h)
- <ctime> (time.h)
- <cuchar> (uchar.h)
- <cwchar> (wchar.h)
- <cwctype> (wctype.h)

Containers:

Input/Output:

library

C library

C Language Library

The C++ library includes the same definitions as the C language library organized in the same structure of header files, with the following differences:

- Each header file has the same name as the C language version but with a "c" prefix and no extension. For example, the C++ equivalent for the C language header file `<stdlib.h>` is `<cstdlib>`.
- Every element of the library is defined within the `std` namespace.

Nevertheless, for compatibility with C, the traditional header names `name.h` (like `stdlib.h`) are also provided with the same definitions within the global namespace. In the examples provided in this reference, this version is used so that the examples are fully C-compatible, although its use is deprecated in C++.

The are also certain specific changes in the C++ implementation:

- `wchar_t`, `char16_t`, `char32_t` and `bool` are fundamental types in C++ and therefore are not defined in the corresponding header where they appear in C. The same applies to several macros in the header <iso646.h>, which are keywords in C++.
- The following functions have changes in their declarations related to the constness of their parameters: strchr, strpbrk, strrchr, strstr, memchr.
- The functions atexit, exit and abort, defined in <cstdlib> have additions to their behavior in C++.
- Overloaded versions of some functions are provided with additional types as parameters and the same semantics, like `float` and `long double` versions of the functions in the cmath header file, or `long` versions for abs and div.

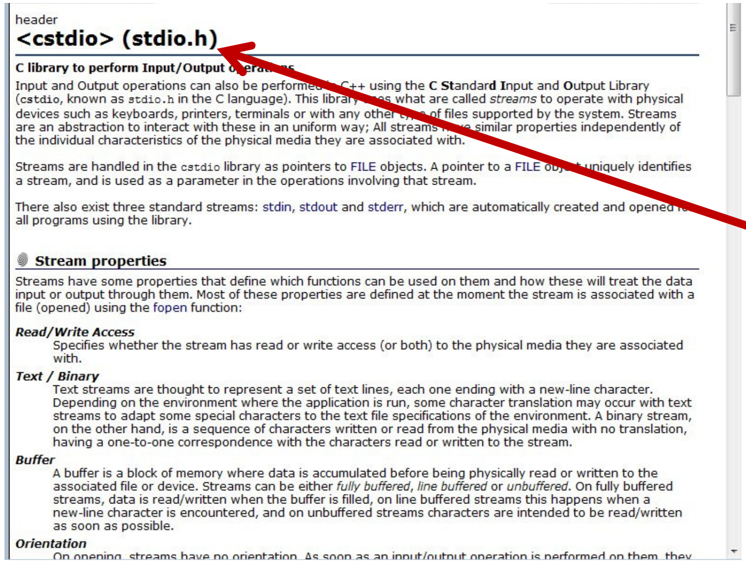
Note on versions

C++98 includes the C library as described by the 1990 ISO C standard and its amendment #1 (ISO/IEC 9899:1990 and ISO/IEC 9899:1990/DAM 1).

C++11 includes the C library as described by the 1999 ISO C standard and its Technical Corrigenda 1, 2 and 3 (ISO/IEC 9899:1999 and ISO/IEC 9899:1999/Cor.1,2,3), plus <cuchar> (as by ISO/IEC 19769:2004).

Other introductions by the 2011 ISO C standard are not compatible with C++.

Headers



header

<cstdio> (stdio.h)

C library to perform Input/Output operations

Input and Output operations can also be performed in C++ using the **C St**andar**d I**nput and **O**utput Library (`cstdio`, known as `stdio.h` in the C language). This library uses what are called *streams* to operate with physical devices such as keyboards, printers, terminals or with any other type of files supported by the system. Streams are an abstraction to interact with these in an uniform way; All streams have similar properties independently of the individual characteristics of the physical media they are associated with.

Streams are handled in the `cstdio` library as pointers to FILE objects. A pointer to a FILE object uniquely identifies a stream, and is used as a parameter in the operations involving that stream.

There also exist three standard streams: stdin, stdout and stderr, which are automatically created and opened for all programs using the library.

Stream properties

Streams have some properties that define which functions can be used on them and how these will treat the data input or output through them. Most of these properties are defined at the moment the stream is associated with a file (opened) using the fopen function:

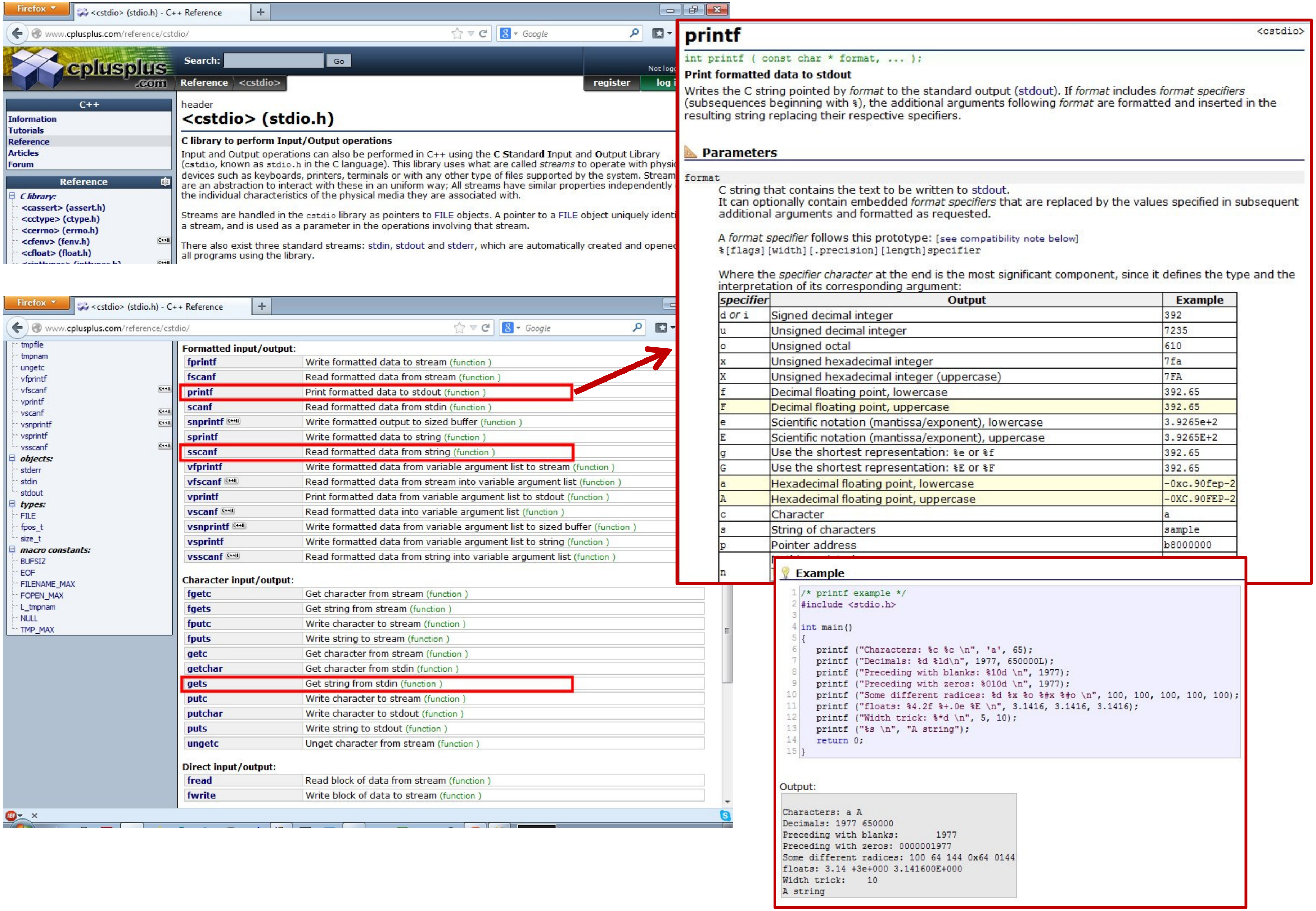
Read/Write Access
Specifies whether the stream has read or write access (or both) to the physical media they are associated with.

Text / Binary
Text streams are thought to represent a set of text lines, each one ending with a new-line character. Depending on the environment where the application is run, some character translation may occur with text streams to adapt some special characters to the text file specifications of the environment. A binary stream, on the other hand, is a sequence of characters written or read from the physical media with no translation, having a one-to-one correspondence with the characters read or written to the stream.

Buffer
A buffer is a block of memory where data is accumulated before being physically read or written to the associated file or device. Streams can be either *fully buffered*, *line buffered* or *unbuffered*. On fully buffered streams, data is read/written when the buffer is filled, on line buffered streams this happens when a new-line character is encountered, and on unbuffered streams characters are intended to be read/written as soon as possible.

Orientation
On opening, streams have no orientation. As soon as an input/output operation is performed on them, they

Treatment of input and output (io) in a program with C language



header

<cstdio> (stdio.h)

C library to perform Input/Output operations

Input and Output operations can also be performed in C++ using the **C** **St**andar**d** **I**nput and **O**utput Library (cstdio, known as stdio.h in the C language). This library uses what are called *streams* to operate with physi devices such as keyboards, printers, terminals or with any other type of files supported by the system. Stream are an abstraction to interact with these in an uniform way; All streams have similar properties independently the individual characteristics of the physical media they are associated with.

Streams are handled in the cstdio library as pointers to FILE objects. A pointer to a FILE object uniquely identi a stream, and is used as a parameter in the operations involving that stream.

There also exist three standard streams: stdin, stdout and stderr, which are automatically created and opened all programs using the library.

Formatted input/output:

fprintf	Write formatted data to stream (function)
fscanf	Read formatted data from stream (function)
printf	Print formatted data to stdout (function)
scanf	Read formatted data from stdin (function)
snprintf C++11	Write formatted output to sized buffer (function)
sprintf	Write formatted data to string (function)
sscanf	Read formatted data from string (function)
vfprintf	Write formatted data from variable argument list to stream (function)
vfscanf C++11	Read formatted data from stream into variable argument list (function)
vprintf	Print formatted data from variable argument list to stdout (function)
vscanf C++11	Read formatted data into variable argument list (function)
vsnprintf C++11	Write formatted data from variable argument list to sized buffer (function)
vsprintf	Write formatted data from variable argument list to string (function)
vsscanf C++11	Read formatted data from string into variable argument list (function)

Character input/output:

fgetc	Get character from stream (function)
fgets	Get string from stream (function)
fputc	Write character to stream (function)
fputs	Write string to stream (function)
getc	Get character from stream (function)
getchar	Get character from stdin (function)
gets	Get string from stdin (function)
putc	Write character to stream (function)
putchar	Write character to stdout (function)
puts	Write string to stdout (function)
ungetc	Unget character from stream (function)

Direct input/output:

fread	Read block of data from stream (function)
fwrite	Write block of data to stream (function)

printf

<cstdio>

```
int printf ( const char * format, ... );
```

Print formatted data to stdout

Writes the C string pointed by *format* to the standard output (stdout). If *format* includes *format specifiers* (subsequences beginning with %), the additional arguments following *format* are formatted and inserted in the resulting string replacing their respective specifiers.

Parameters

format

C string that contains the text to be written to stdout.
It can optionally contain embedded *format specifiers* that are replaced by the values specified in subsequent additional arguments and formatted as requested.

A *format specifier* follows this prototype: [see compatibility note below]
%[flags][width][.precision][length]specifier

Where the *specifier character* at the end is the most significant component, since it defines the type and the interpretation of its corresponding argument:

specifier	Output	Example
d *or* i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n		

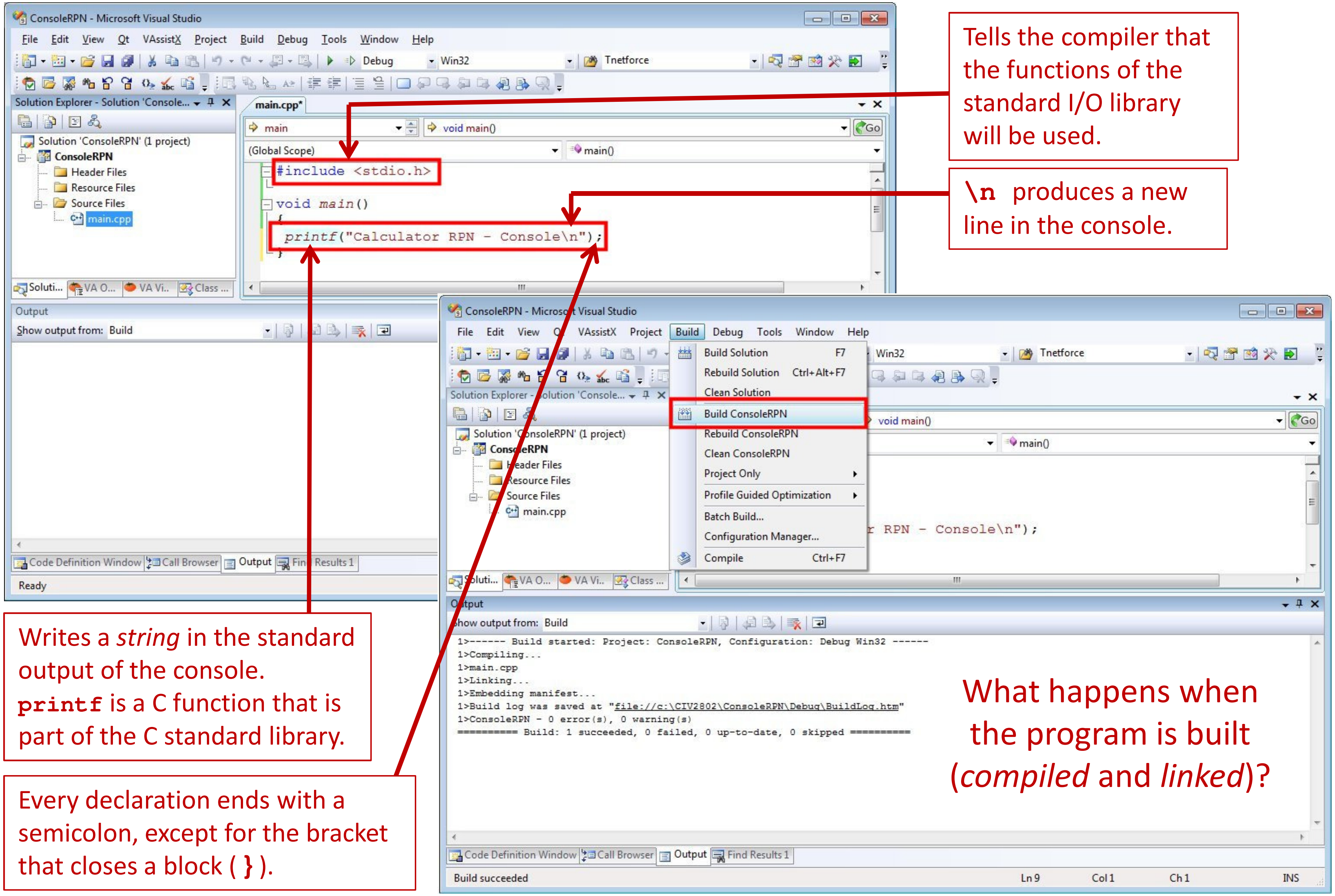
Example

```
/* printf example */
#include <stdio.h>

int main()
{
   printf ("Characters: %c %c \n", 'a', 65);
   printf ("Decimals: %d %ld\n", 1977, 650000L);
   printf ("Preceding with blanks: %10d \n", 1977);
   printf ("Preceding with zeros: %010d \n", 1977);
   printf ("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
   printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
   printf ("Width trick: %*d \n", 5, 10);
   printf ("%s \n", "A string");
   return 0;
}
```

Output:

```
Characters: a A
Decimals: 1977 650000
Preceding with blanks:       1977
Preceding with zeros: 0000001977
Some different radices: 100 64 144 0x64 0144
floats: 3.14 +3e+000 3.141600E+000
Width trick:    10
A string
```

Printing a message in the console using the C standard I/O library



```
#include <stdio.h>

void main()
{
  printf("Calculator RPN - Console\n");
}
```

```
1>------ Build started: Project: ConsoleRPN, Configuration: Debug Win32 ------
1>Compiling...
1>main.cpp
1>Linking...
1>Embedding manifest...
1>Build log was saved at "file://c:\CIV2802\ConsoleRPN\Debug\BuildLog.htm"
1>ConsoleRPN - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Tells the compiler that the functions of the standard I/O library will be used.

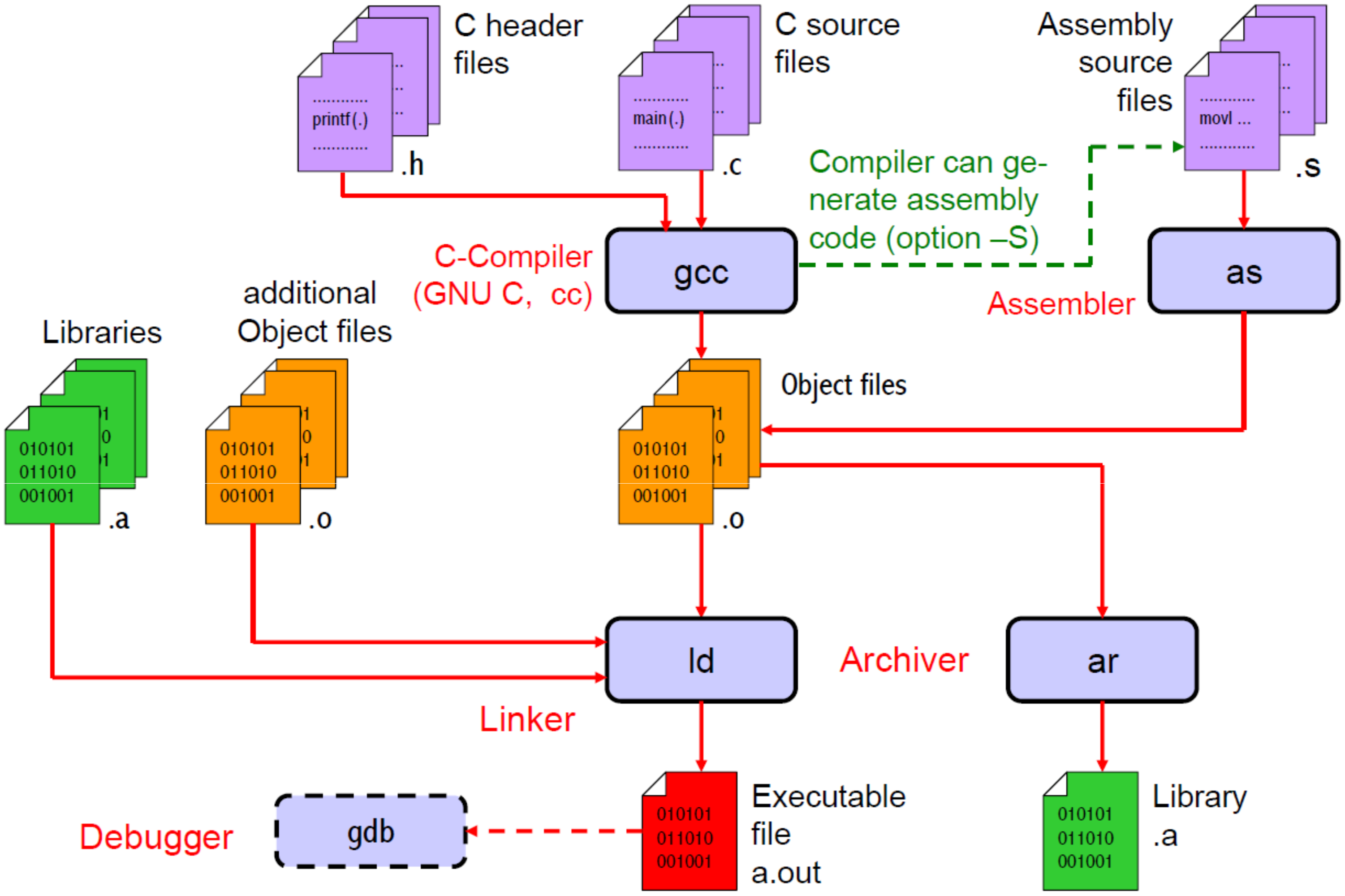
\n produces a new line in the console.

Writes a *string* in the standard output of the console. **printf** is a C function that is part of the C standard library.

Every declaration ends with a semicolon, except for the bracket that closes a block (**}**).

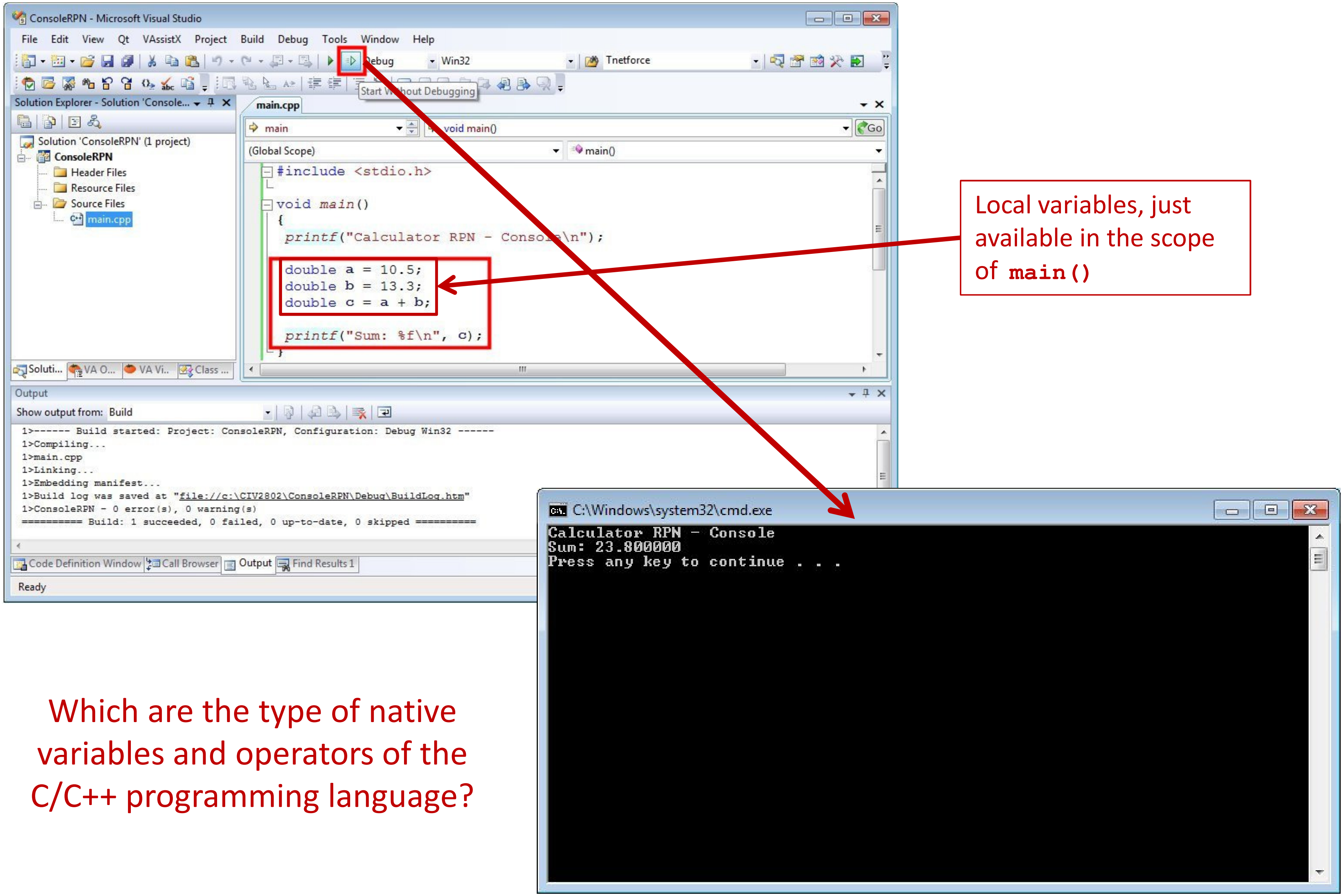
What happens when the program is built (*compiled* and *linked*)?

From C codes to an executable binary file



Source: René Müller, Introduction to the C-Language and Programming Environment, Winter Semester 2005/06

Printing to the console (output) the sum of two numbers



```
#include <stdio.h>

void main()
{
 printf("Calculator RPN - Console\n");

 double a = 10.5;
 double b = 13.3;
 double c = a + b;

 printf("Sum: %f\n", c);
}
```

```
1>------ Build started: Project: ConsoleRPN, Configuration: Debug Win32 ------
1>Compiling...
1>main.cpp
1>Linking...
1>Embedding manifest...
1>Build log was saved at "file://c:\CIV2802\ConsoleRPN\Debug\BuildLog.htm"
1>ConsoleRPN - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

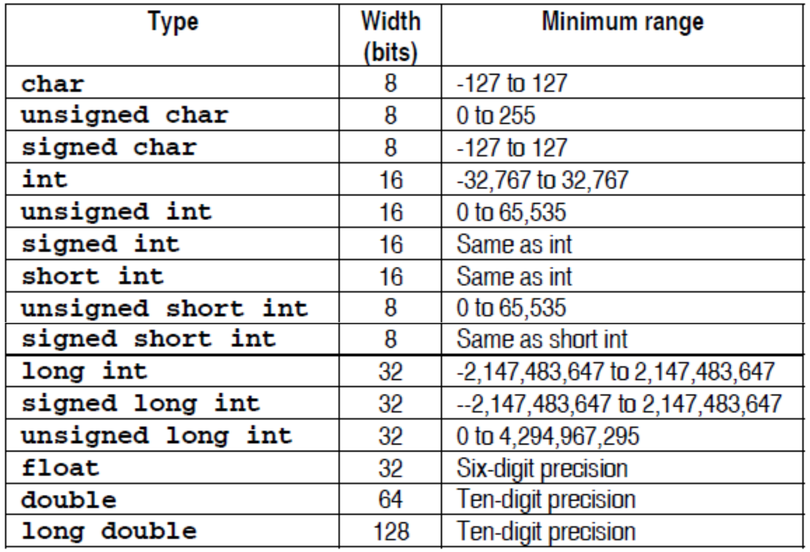
```
Calculator RPN - Console
Sum: 23.800000
Press any key to continue . . .
```

Local variables, just available in the scope of **`main()`**

Which are the type of native variables and operators of the C/C++ programming language?

C data types

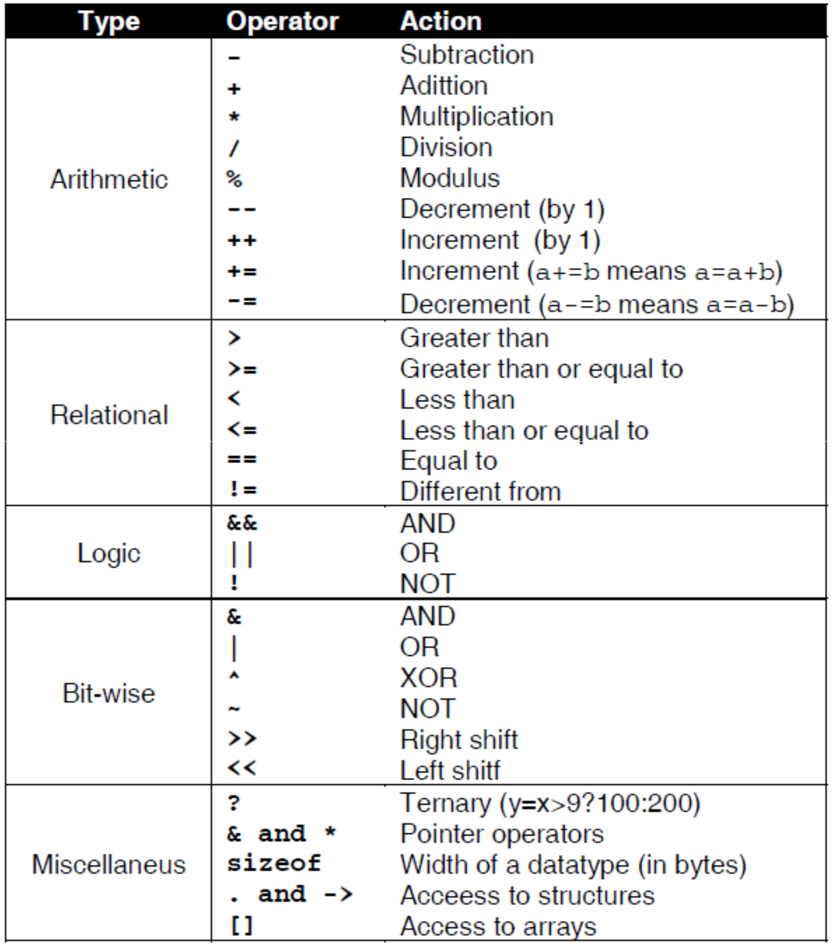
- **Four basic data types**
  - **`char`**: character
  - **`int`**: integer
  - **`float`**: real or floating point
  - **`double`**: double precision float
- **Four modifiers**
  - **`signed`**
  - **`unsigned`**
  - **`long`**
  - **`short`**
- **Four storage classes**
  - **`auto`**: variable is not required outside its block (the default)
  - **`register`**: the variable will be allocated on a CPU register
  - **`static`**: allows a local variable to retain its previous value upon reentry
  - **`extern`**: global variable declared in another file
- **Additionally, C supports**
  - the null data type: **`void`**
  - Any user-defined types



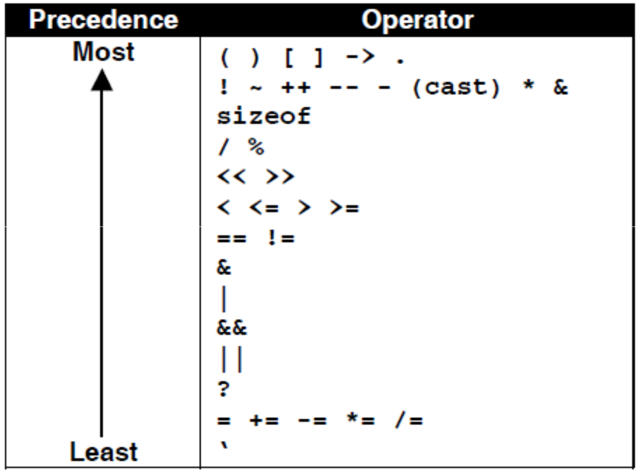
Type	Width (bits)	Minimum range
`char`	8	-127 to 127
`unsigned char`	8	0 to 255
`signed char`	8	-127 to 127
`int`	16	-32,767 to 32,767
`unsigned int`	16	0 to 65,535
`signed int`	16	Same as int
`short int`	16	Same as int
`unsigned short int`	8	0 to 65,535
`signed short int`	8	Same as short int
`long int`	32	-2,147,483,647 to 2,147,483,647
`signed long int`	32	--2,147,483,647 to 2,147,483,647
`unsigned long int`	32	0 to 4,294,967,295
`float`	32	Six-digit precision
`double`	64	Ten-digit precision
`long double`	128	Ten-digit precision

Source: Ricardo Gutierrez-Osuna, *Microprocessor-based System Design*,
Wright State University.

C operators



Type	Operator	Action
Arithmetic	`-`	Subtraction
	`+`	Adittion
	`*`	Multiplication
	`/`	Division
	`%`	Modulus
	`--`	Decrement (by 1)
	`++`	Increment (by 1)
	`+=`	Increment (`a+=b` means `a=a+b`)
	`-=`	Decrement (`a-=b` means `a=a-b`)
Relational	`>`	Greater than
	`>=`	Greater than or equal to
	`<`	Less than
	`<=`	Less than or equal to
	`==`	Equal to
	`!=`	Different from
Logic	`&&`	AND
	`||`	OR
	`!`	NOT
Bit-wise	`&`	AND
	`|`	OR
	`^`	XOR
	`~`	NOT
	`>>`	Right shift
	`<<`	Left shiff
Miscellaneus	`?`	Ternary (y=x>9?100:200)
	`&` **and** `*`	Pointer operators
	`sizeof`	Width of a datatype (in bytes)
	`.` **and** `->`	Acceess to structures
	`[]`	Access to arrays

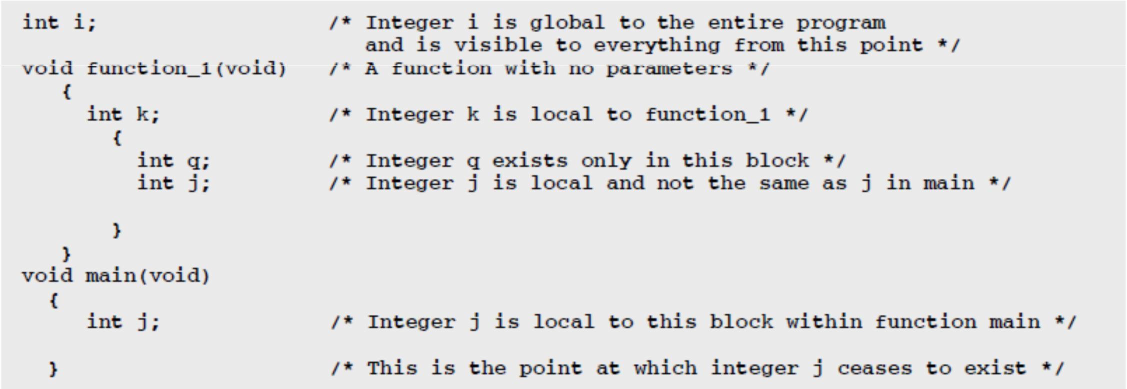


Precedence	Operator
Most	`( ) [ ] -> .`
	`! ~ ++ -- - (cast) * & sizeof`
	`/ %`
	`<< >>`
	`< <= > >=`
	`== !=`
	`&`
	`|`
	`&&`
	`||`
	`?`
	`= += -= *= /=`
Least	`` ` ``

Source: Ricardo Gutierrez-Osuna, *Microprocessor-based System Design*, Wright State University.

Variable declaration and scope

- **Variables <u>MUST</u> be declared before they are used**
  - Any declaration MUST precede the first statement in a block
- **Variables declared inside a block are local to that block**
  - They cannot be accessed from outside the block
- **Variables can be initialized when they are declared of afterwards**

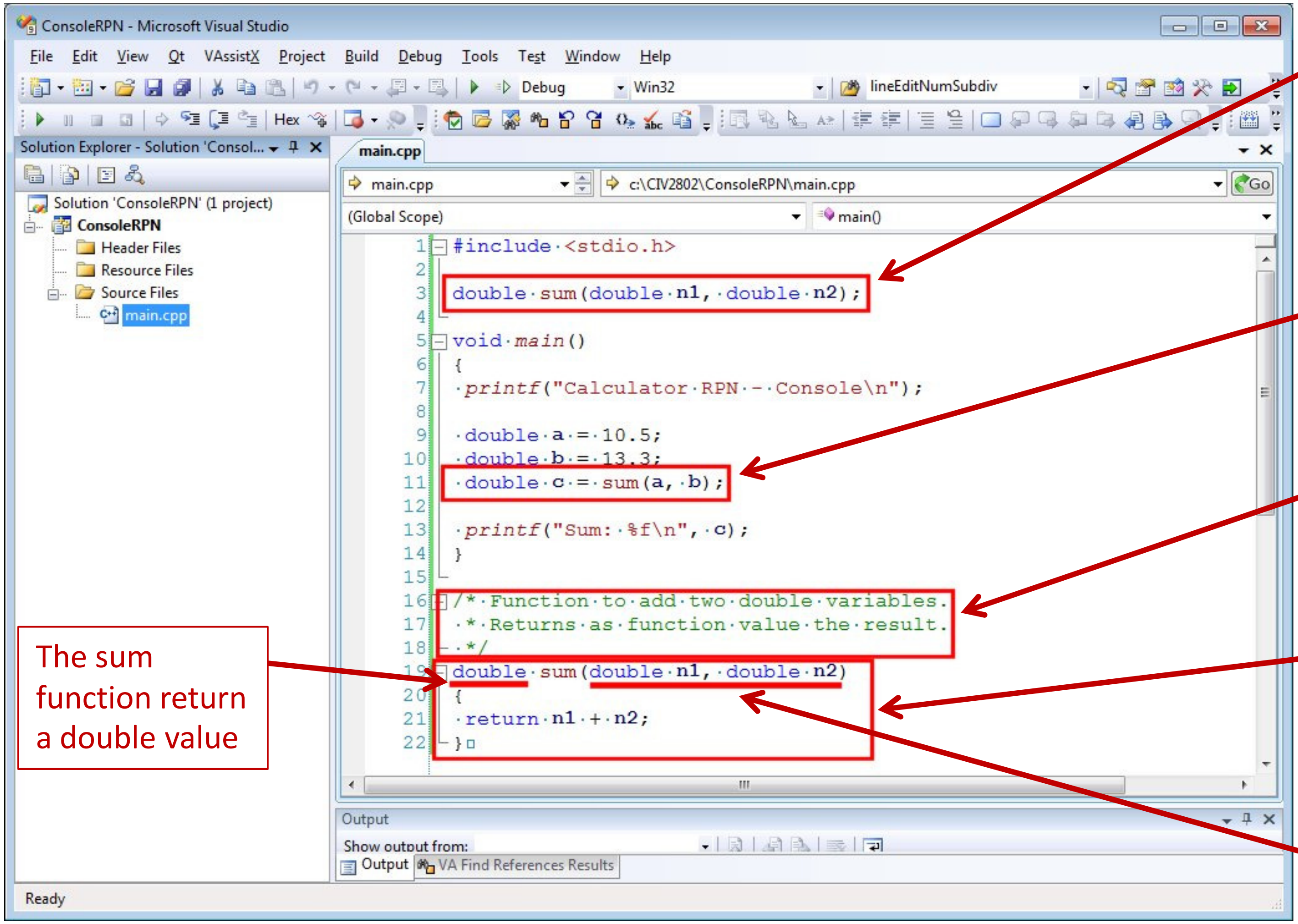


```
int i;                    /* Integer i is global to the entire program
                             and is visible to everything from this point */
void function_1(void)     /* A function with no parameters */
   {
     int k;               /* Integer k is local to function_1 */
       {
         int q;           /* Integer q exists only in this block */
         int j;           /* Integer j is local and not the same as j in main */

       }
   }
void main(void)
  {
     int j;               /* Integer j is local to this block within function main */

  }                       /* This is the point at which integer j ceases to exist */
```

Function (*function*)? What is this?
How to perform a sum operation using a function?

Source: Ricardo Gutierrez-Osuna, *Microprocessor-based System Design*,
Wright State University.

Implementando uma função para executar a soma de dois números


ConsoleRPN - Microsoft Visual Studio
File Edit View Qt VAssistX Project Build Debug Tools Test Window Help
Debug
Win32
lineEditNumSubdiv
Hex
Solution Explorer - Solution 'Consol...
Solution 'ConsoleRPN' (1 project)
ConsoleRPN
Header Files
Resource Files
Source Files
main.cpp
main.cpp
c:\CIV2802\ConsoleRPN\main.cpp
Go
(Global Scope)
main()
Output
Show output from:
Output
VA Find References Results
Ready

```c
#include <stdio.h>

double sum(double n1, double n2);

void main()
{
 printf("Calculator RPN - Console\n");

 double a = 10.5;
 double b = 13.3;
 double c = sum(a, b);

 printf("Sum: %f\n", c);
}

/* Function to add two double variables.
 * Returns as function value the result.
 */
double sum(double n1, double n2)
{
 return n1 + n2;
}
```

Declaration (*prototype*) of the **sum** funtion

Calls the **sum** function

A pair of /* */ defines a comment that is ignored by the compiler

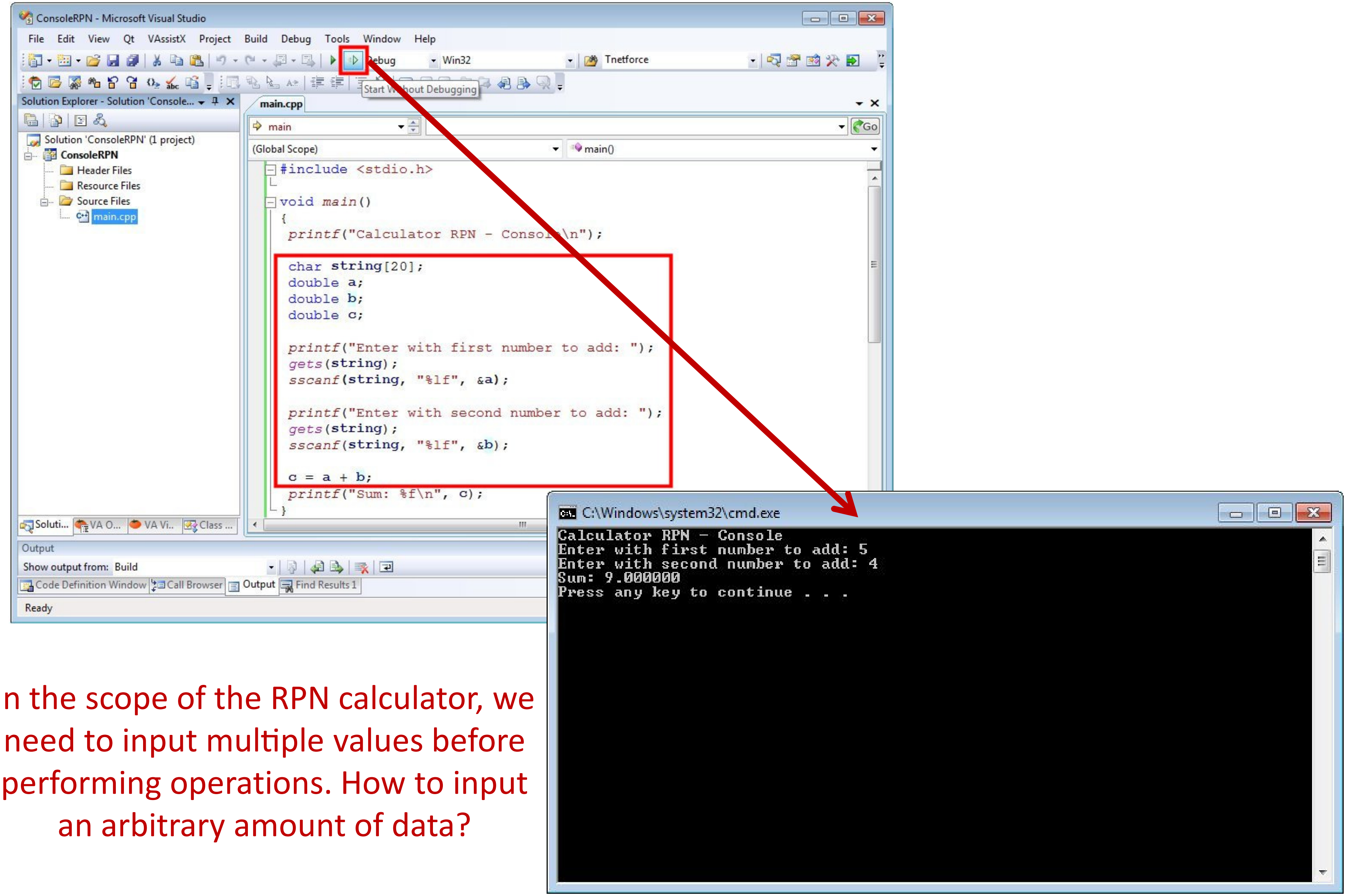
Implementation of the **sum** function

The **sum** function receives two values of double type in the list of parameters

The sum function return a double value

How to input data in C/C++ language?

Inserting by the console (input) the two number to be added



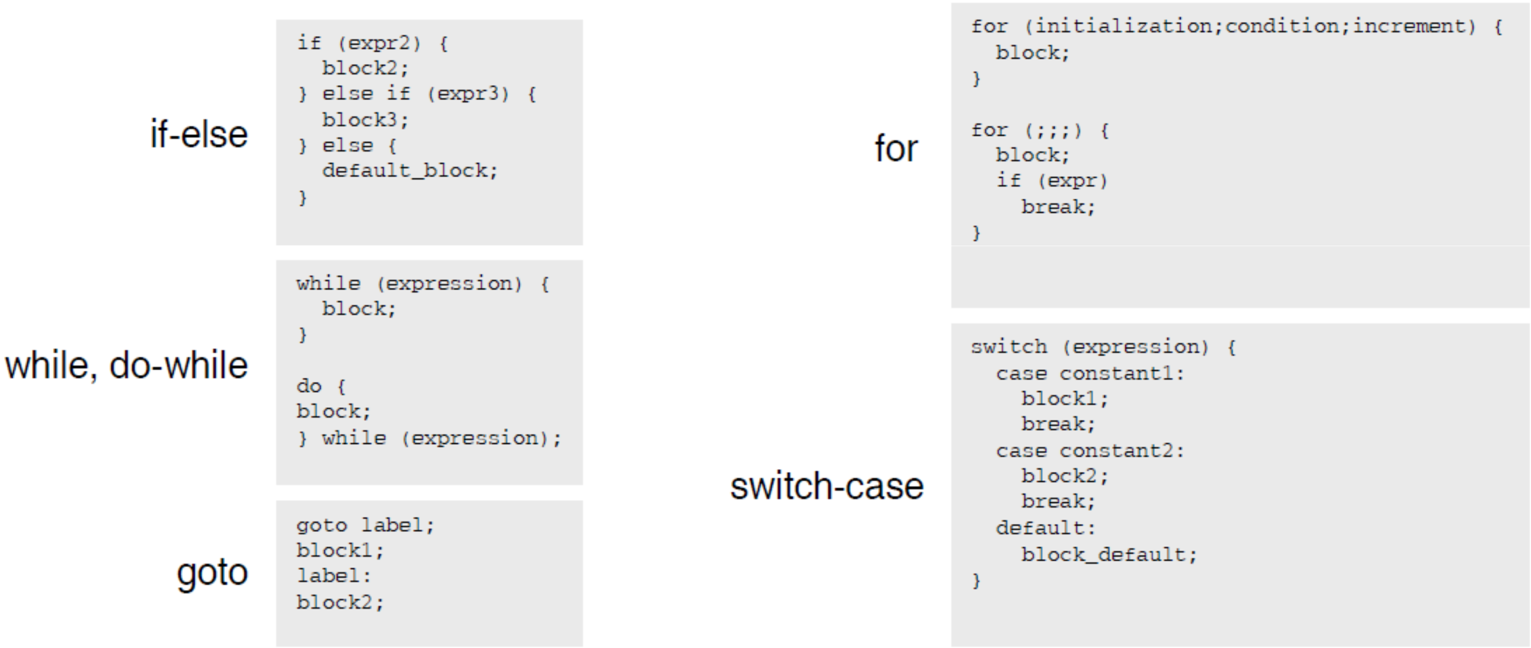
```
#include <stdio.h>

void main()
{
 printf("Calculator RPN - Console\n");

 char string[20];
 double a;
 double b;
 double c;

 printf("Enter with first number to add: ");
 gets(string);
 sscanf(string, "%lf", &a);

 printf("Enter with second number to add: ");
 gets(string);
 sscanf(string, "%lf", &b);

 c = a + b;
 printf("Sum: %f\n", c);
}
```

```
Calculator RPN - Console
Enter with first number to add: 5
Enter with second number to add: 4
Sum: 9.000000
Press any key to continue . . .
```

In the scope of the RPN calculator, we need to input multiple values before performing operations. How to input an arbitrary amount of data?

Loops and iterations

- **In C any expression different than ZERO is `TRUE`, including negative numbers, strings, …**
- **C provides the following constructs**



if-else

```
if (expr2) {
  block2;
} else if (expr3) {
  block3;
} else {
  default_block;
}
```

while, do-while

```
while (expression) {
  block;
}

do {
block;
} while (expression);
```

goto

```
goto label;
block1;
label:
block2;
```

for

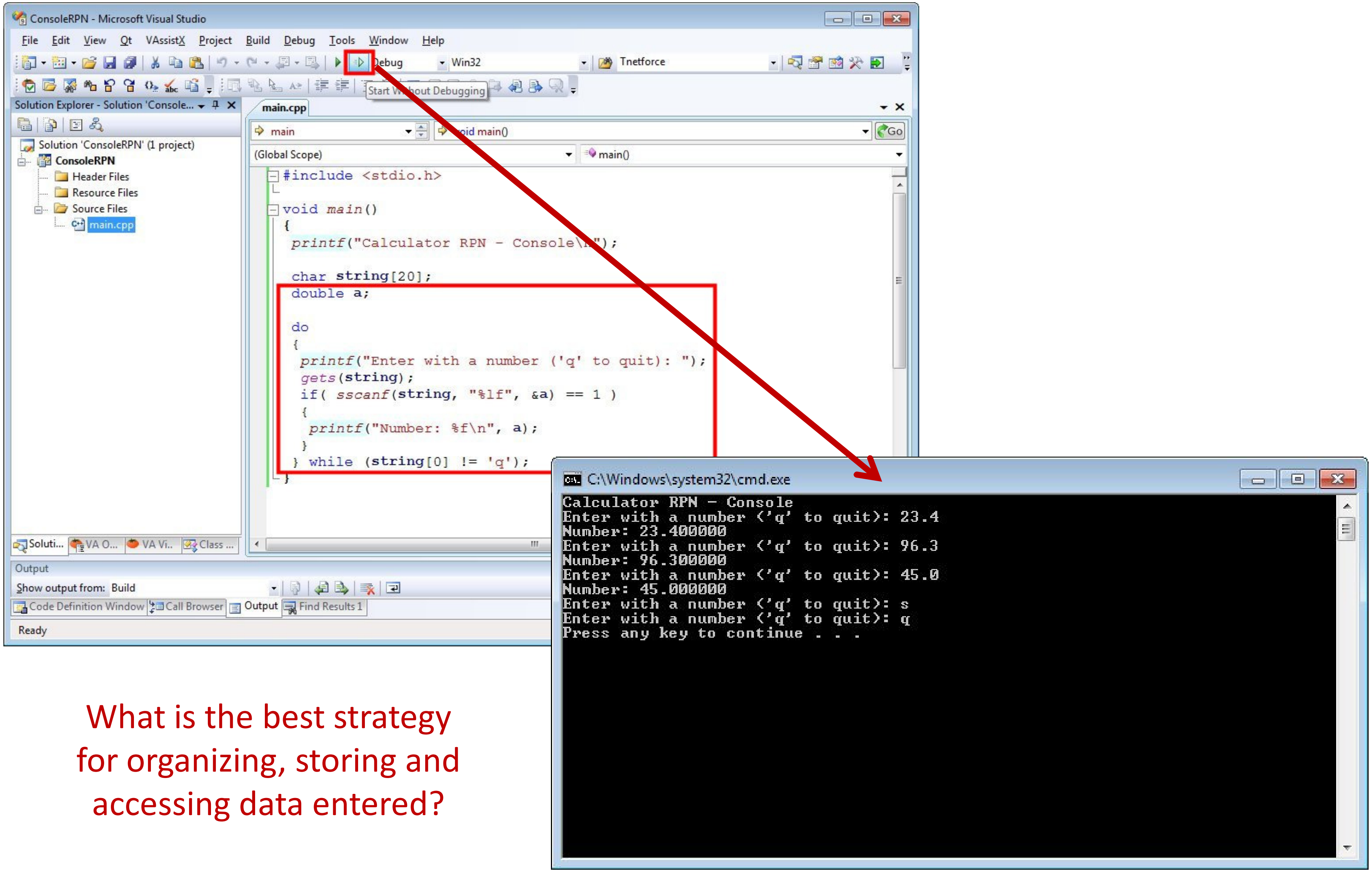
```
for (initialization;condition;increment) {
  block;
}

for (;;;) {
  block;
  if (expr)
    break;
}
```

switch-case

```
switch (expression) {
  case constant1:
    block1;
    break;
  case constant2:
    block2;
    break;
  default:
    block_default;
}
```

Source: Ricardo Gutierrez-Osuna, *Microprocessor-based System Design,*
Wright State University.

Preparing the program to insert any amount of numbers, and definition of a command to quit the program

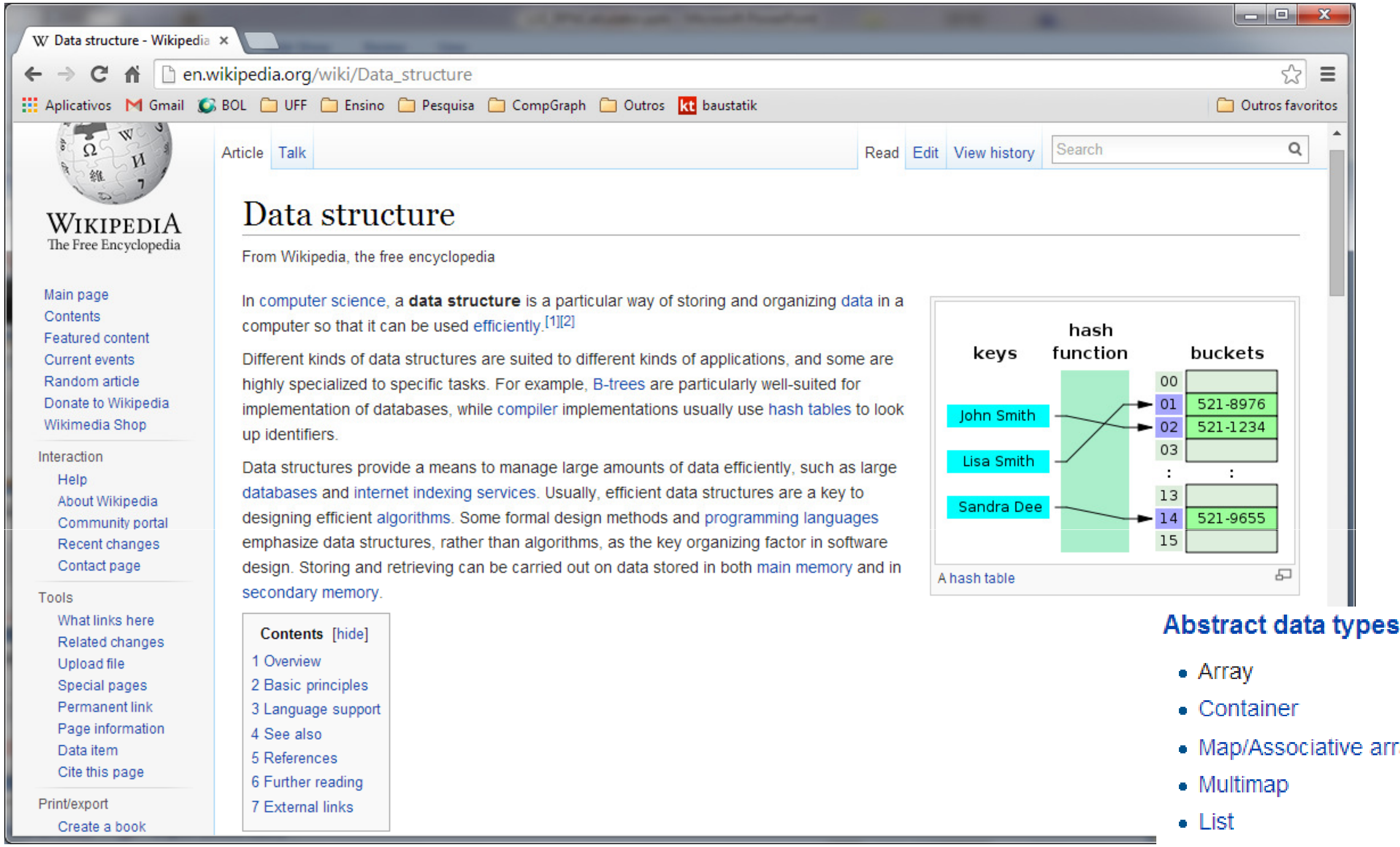

ConsoleRPN - Microsoft Visual Studio
File Edit View Qt VAssistX Project Build Debug Tools Window Help
Debug
Win32
Tnetforce
Start Without Debugging
Solution Explorer - Solution 'Console...
Solution 'ConsoleRPN' (1 project)
ConsoleRPN
Header Files
Resource Files
Source Files
main.cpp
main
void main()
Go
(Global Scope)
main()
Soluti... VA O... VA Vi... Class ...
Output
Show output from: Build
Code Definition Window
Call Browser
Output
Find Results 1
Ready
C:\Windows\system32\cmd.exe

```
#include <stdio.h>

void main()
{
 printf("Calculator RPN - Console\n");

 char string[20];
 double a;

 do
 {
  printf("Enter with a number ('q' to quit): ");
  gets(string);
  if( sscanf(string, "%lf", &a) == 1 )
  {
   printf("Number: %f\n", a);
  }
 } while (string[0] != 'q');
}
```

```
Calculator RPN - Console
Enter with a number ('q' to quit): 23.4
Number: 23.400000
Enter with a number ('q' to quit): 96.3
Number: 96.300000
Enter with a number ('q' to quit): 45.0
Number: 45.000000
Enter with a number ('q' to quit): s
Enter with a number ('q' to quit): q
Press any key to continue . . .
```

What is the best strategy for organizing, storing and accessing data entered?

Data Structures



Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikimedia Shop

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Data item
Cite this page

Print/export
Create a book

Data structure

From Wikipedia, the free encyclopedia

In computer science, a **data structure** is a particular way of storing and organizing data in a computer so that it can be used efficiently.[1][2]

Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, B-trees are particularly well-suited for implementation of databases, while compiler implementations usually use hash tables to look up identifiers.

Data structures provide a means to manage large amounts of data efficiently, such as large databases and internet indexing services. Usually, efficient data structures are a key to designing efficient algorithms. Some formal design methods and programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design. Storing and retrieving can be carried out on data stored in both main memory and in secondary memory.

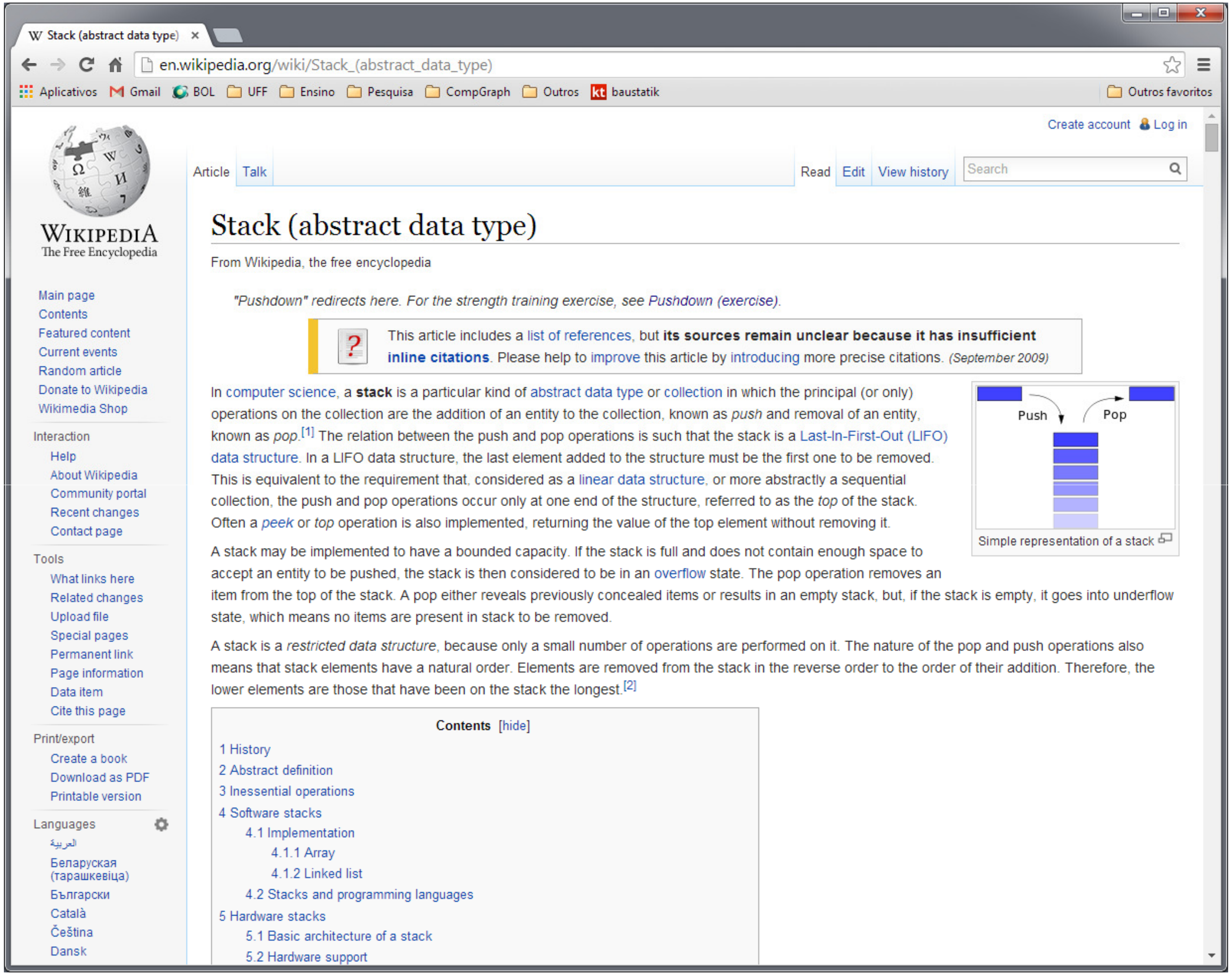
A hash table

Abstract data types [edit]

- Array
- Container
- Map/Associative array/Dictionary
- Multimap
- List
- Set
- Multiset
- Priority queue
- Queue
- Deque
- Stack
- String
- Tree
- Graph

What is the most suitable data structure for the problem of RPN calculator?

The Stack Data Structure



Stack (abstract data type)

From Wikipedia, the free encyclopedia

"Pushdown" redirects here. For the strength training exercise, see Pushdown (exercise).

This article includes a list of references, but **its sources remain unclear because it has insufficient inline citations**. Please help to improve this article by introducing more precise citations. *(September 2009)*

In computer science, a **stack** is a particular kind of abstract data type or collection in which the principal (or only) operations on the collection are the addition of an entity to the collection, known as *push* and removal of an entity, known as *pop*.[1] The relation between the push and pop operations is such that the stack is a Last-In-First-Out (LIFO) data structure. In a LIFO data structure, the last element added to the structure must be the first one to be removed. This is equivalent to the requirement that, considered as a linear data structure, or more abstractly a sequential collection, the push and pop operations occur only at one end of the structure, referred to as the *top* of the stack. Often a *peek* or *top* operation is also implemented, returning the value of the top element without removing it.

Simple representation of a stack

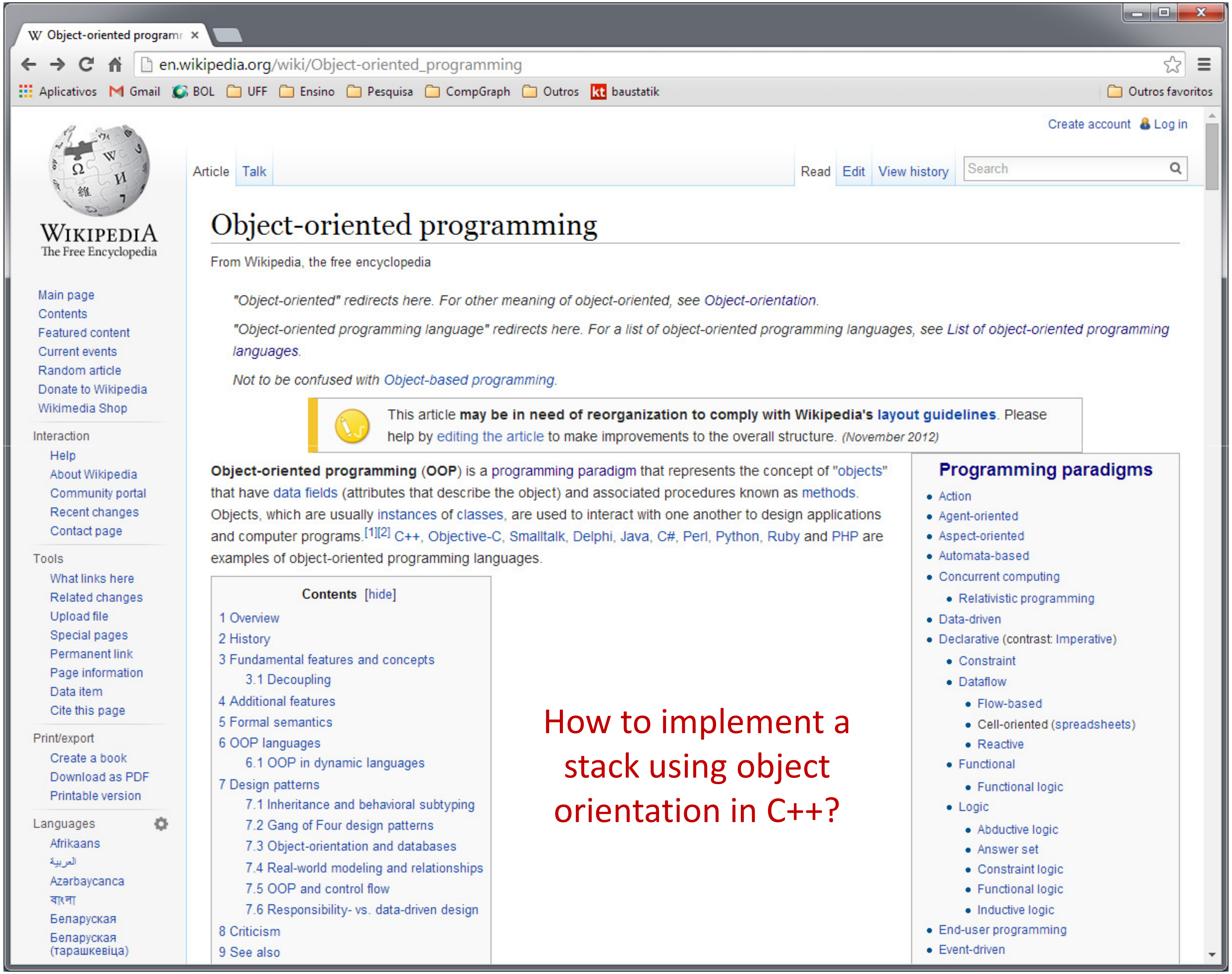
A stack may be implemented to have a bounded capacity. If the stack is full and does not contain enough space to accept an entity to be pushed, the stack is then considered to be in an overflow state. The pop operation removes an item from the top of the stack. A pop either reveals previously concealed items or results in an empty stack, but, if the stack is empty, it goes into underflow state, which means no items are present in stack to be removed.

A stack is a *restricted data structure*, because only a small number of operations are performed on it. The nature of the pop and push operations also means that stack elements have a natural order. Elements are removed from the stack in the reverse order to the order of their addition. Therefore, the lower elements are those that have been on the stack the longest.[2]

Contents [hide]

How to implement a stack data structure using the C/C++ language?

Object Oriented Programming



Object-oriented programming

From Wikipedia, the free encyclopedia

"Object-oriented" redirects here. For other meaning of object-oriented, see Object-orientation.

"Object-oriented programming language" redirects here. For a list of object-oriented programming languages, see List of object-oriented programming languages.

Not to be confused with Object-based programming.

This article **may be in need of reorganization to comply with Wikipedia's layout guidelines**. Please help by editing the article to make improvements to the overall structure. *(November 2012)*

Object-oriented programming (**OOP**) is a programming paradigm that represents the concept of "objects" that have data fields (attributes that describe the object) and associated procedures known as methods. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.[1][2] C++, Objective-C, Smalltalk, Delphi, Java, C#, Perl, Python, Ruby and PHP are examples of object-oriented programming languages.

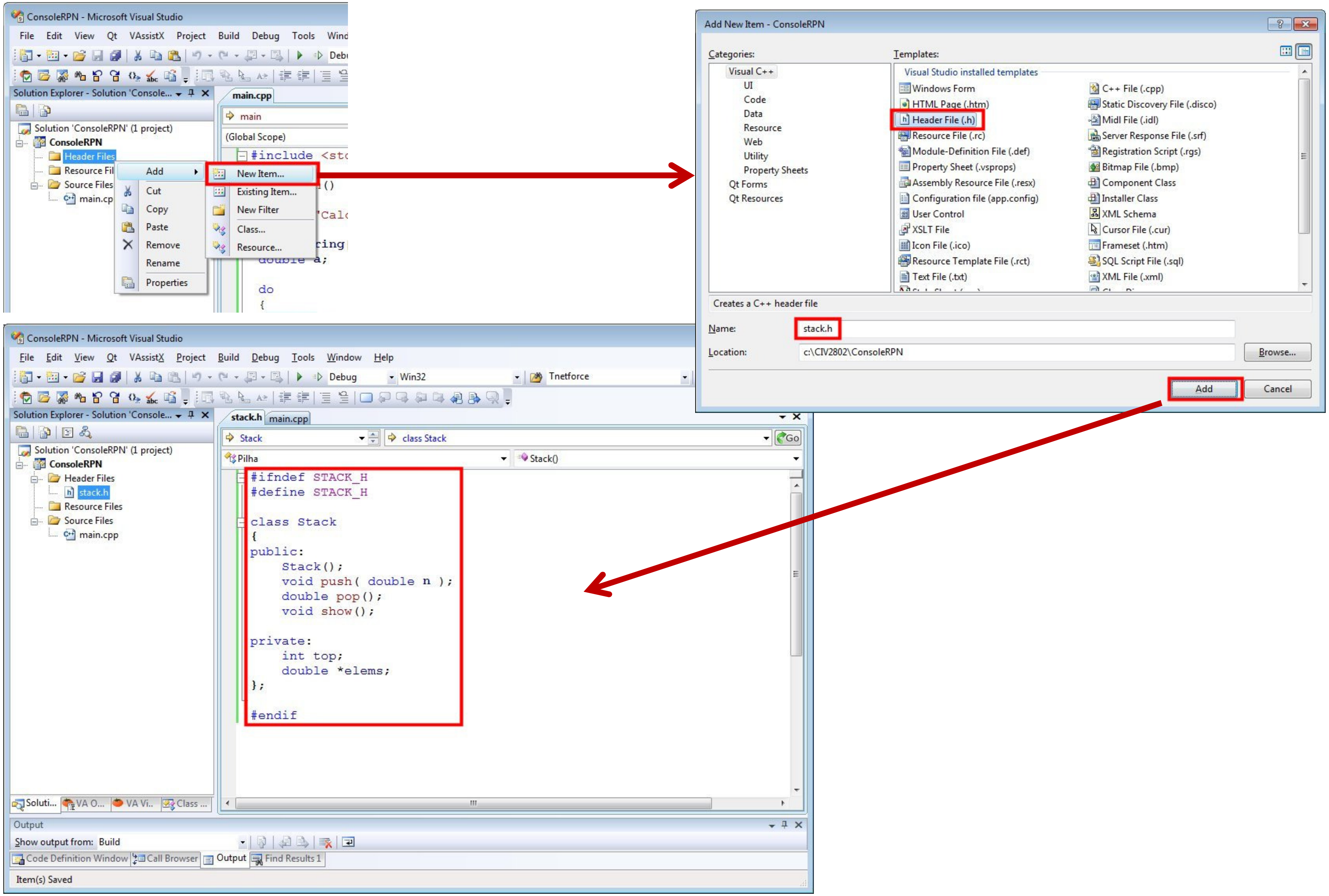
Contents [hide]

Programming paradigms

- Action
- Agent-oriented
- Aspect-oriented
- Automata-based
- Concurrent computing
  - Relativistic programming
- Data-driven
- Declarative (contrast: Imperative)
  - Constraint
  - Dataflow
    - Flow-based
    - Cell-oriented (spreadsheets)
    - Reactive
  - Functional
    - Functional logic
  - Logic
    - Abductive logic
    - Answer set
    - Constraint logic
    - Functional logic
    - Inductive logic
- End-user programming
- Event-driven

How to implement a stack using object orientation in C++?

Implementing a *Stack*. Creating the file: “stack.h”



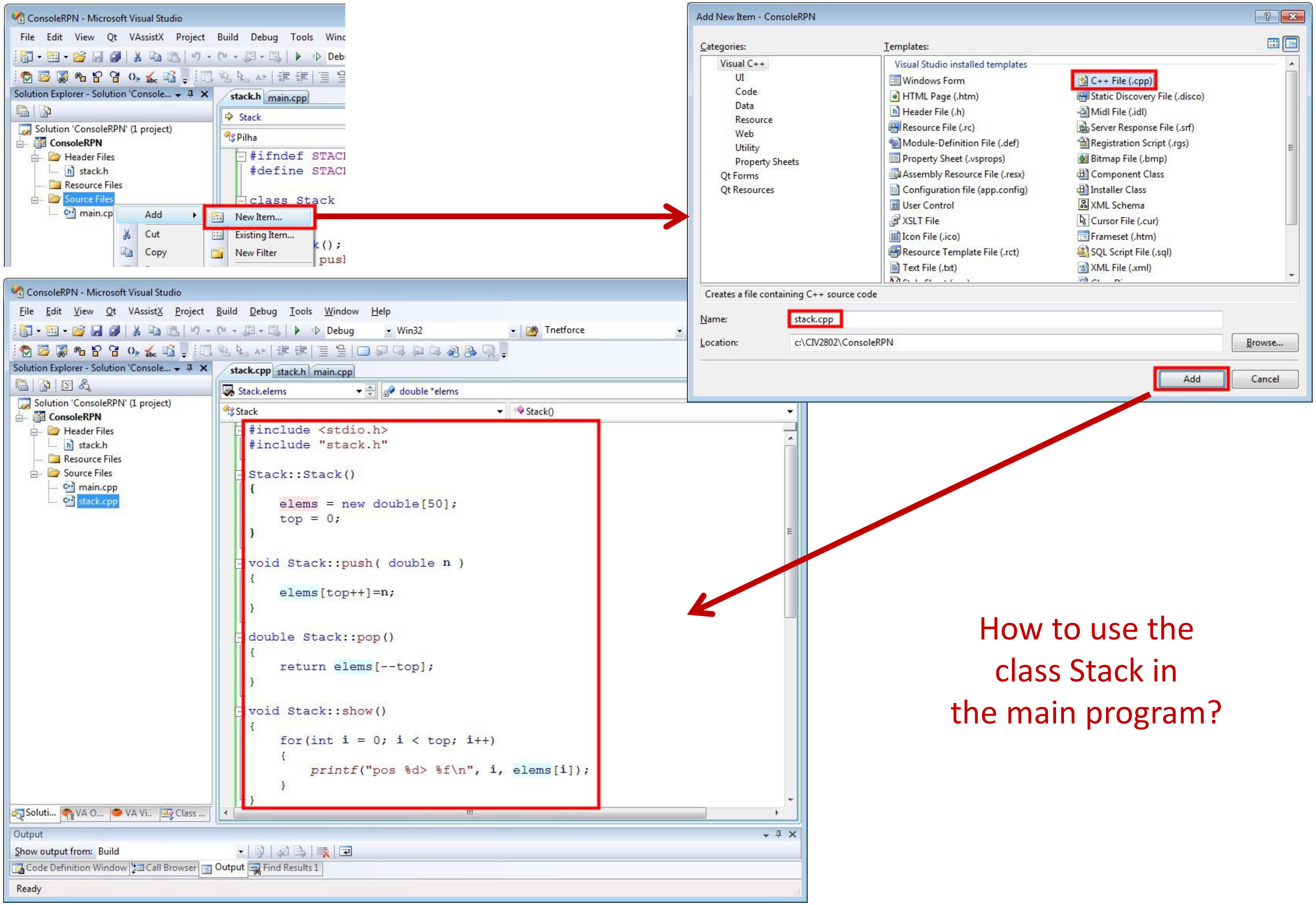
Creates a C++ header file

Name: stack.h

Location: c:\CIV2802\ConsoleRPN

```
#ifndef STACK_H
#define STACK_H

class Stack
{
public:
      Stack();
      void push( double n );
      double pop();
      void show();

private:
      int top;
      double *elems;
};

#endif
```

Implementing a *Stack*. Creating the file: “stack.cpp”


ConsoleRPN - Microsoft Visual Studio
File Edit View Qt VAssistX Project Build Debug Tools Wind
Solution Explorer - Solution 'Console...
stack.h
main.cpp
Stack
Pilha
Solution 'ConsoleRPN' (1 project)
ConsoleRPN
Header Files
stack.h
Resource Files
Source Files
main.cp
Add
Cut
Copy
New Item...
Existing Item...
New Filter

```
#ifndef STACI
#define STACI

class Stack
```

Creates a file containing C++ source code

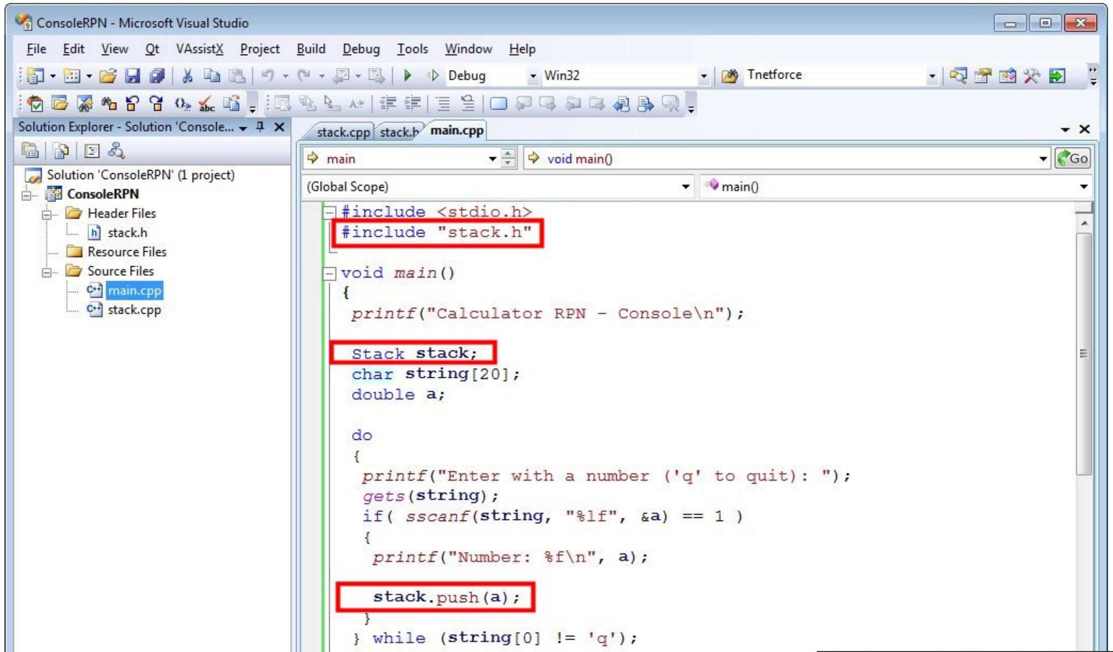
Name: stack.cpp

Location: c:\CIV2802\ConsoleRPN

```
#include <stdio.h>
#include "stack.h"

Stack::Stack()
{
    elems = new double[50];
    top = 0;
}

void Stack::push( double n )
{
    elems[top++]=n;
}

double Stack::pop()
{
    return elems[--top];
}

void Stack::show()
{
    for(int i = 0; i < top; i++)
    {
        printf("pos %d> %f\n", i, elems[i]);
    }
}
```

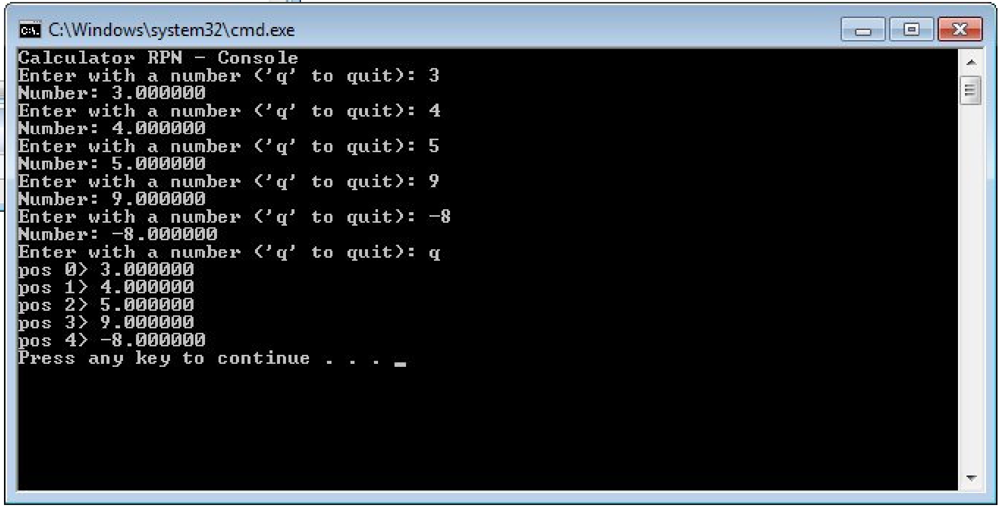
How to use the class Stack in the main program?

Using the Stack class in the main program to store the data



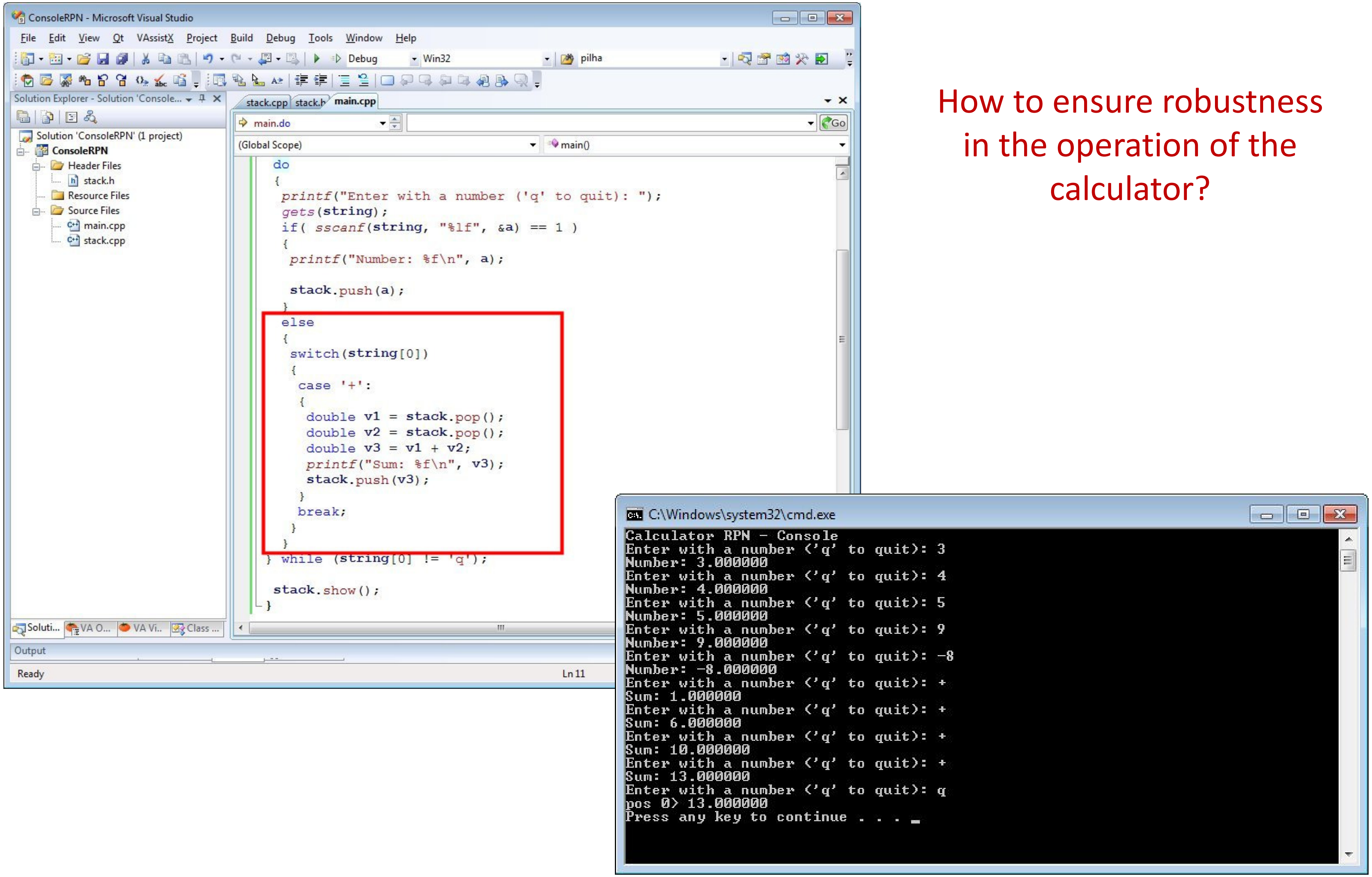
```
#include <stdio.h>
#include "stack.h"

void main()
{
 printf("Calculator RPN - Console\n");

 Stack stack;
 char string[20];
 double a;

 do
 {
  printf("Enter with a number ('q' to quit): ");
  gets(string);
  if( sscanf(string, "%lf", &a) == 1 )
  {
   printf("Number: %f\n", a);

   stack.push(a);
  }
 } while (string[0] != 'q');

 stack.show();
}
```

How to use an object of the Stack class to perform the operations with the calculator?



```
Calculator RPN - Console
Enter with a number ('q' to quit): 3
Number: 3.000000
Enter with a number ('q' to quit): 4
Number: 4.000000
Enter with a number ('q' to quit): 5
Number: 5.000000
Enter with a number ('q' to quit): 9
Number: 9.000000
Enter with a number ('q' to quit): -8
Number: -8.000000
Enter with a number ('q' to quit): q
pos 0> 3.000000
pos 1> 4.000000
pos 2> 5.000000
pos 3> 9.000000
pos 4> -8.000000
Press any key to continue . . . _
```

Implementation of addition operation using the stack data



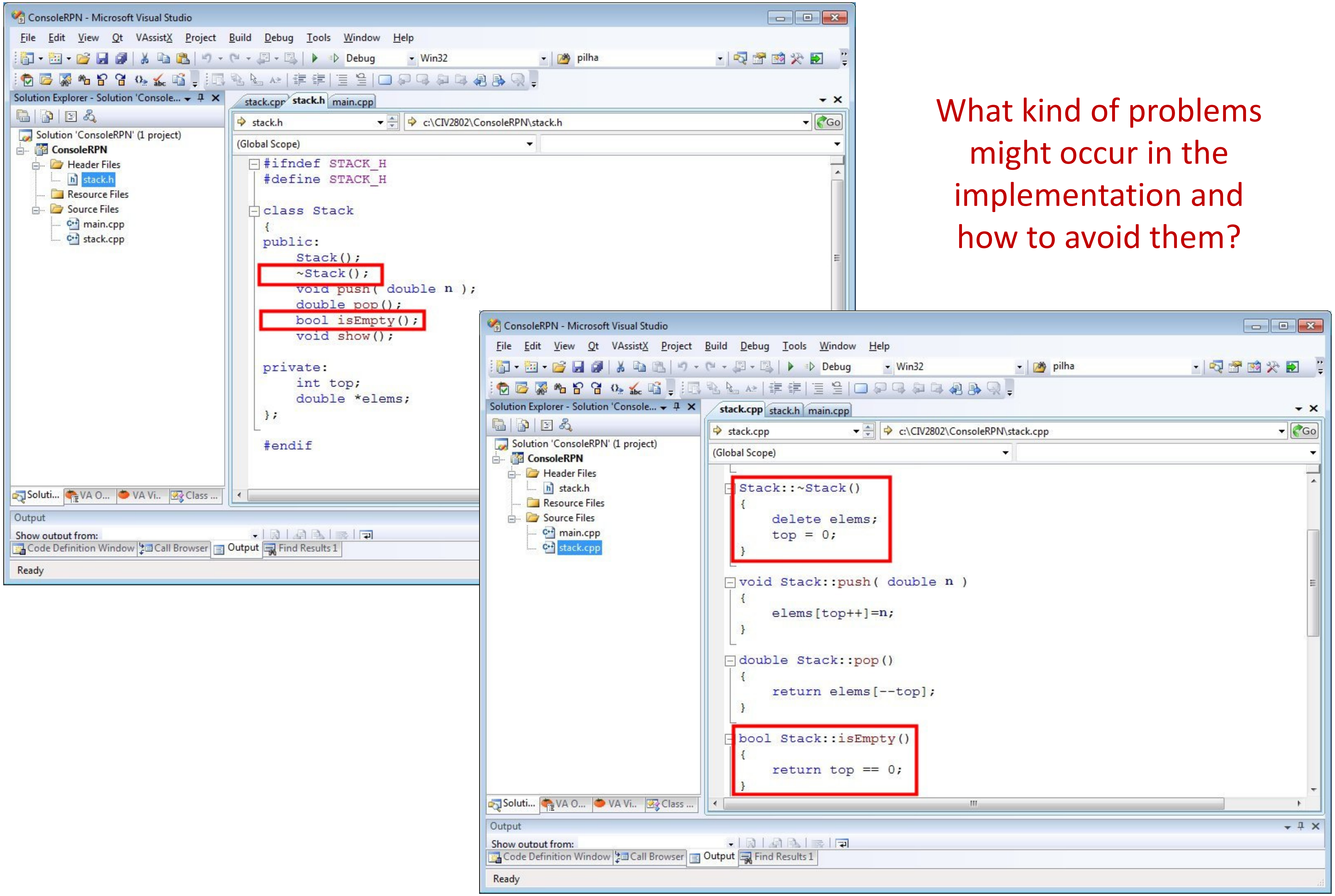
ConsoleRPN - Microsoft Visual Studio
File Edit View Qt VAssistX Project Build Debug Tools Window Help
Debug
Win32
pilha
Solution Explorer - Solution 'Console...
Solution 'ConsoleRPN' (1 project)
ConsoleRPN
Header Files
stack.h
Resource Files
Source Files
main.cpp
stack.cpp
stack.cpp
stack.h
main.cpp
main.do
Go
(Global Scope)
main()
Soluti...
VA O...
VA Vi...
Class ...
Output
Ready
Ln 11
C:\Windows\system32\cmd.exe

```
do
{
 printf("Enter with a number ('q' to quit): ");
 gets(string);
 if( sscanf(string, "%lf", &a) == 1 )
 {
  printf("Number: %f\n", a);

  stack.push(a);
 }
 else
 {
  switch(string[0])
  {
   case '+':
   {
    double v1 = stack.pop();
    double v2 = stack.pop();
    double v3 = v1 + v2;
    printf("Sum: %f\n", v3);
    stack.push(v3);
   }
   break;
  }
 }
} while (string[0] != 'q');

stack.show();
}
```

```
Calculator RPN - Console
Enter with a number ('q' to quit): 3
Number: 3.000000
Enter with a number ('q' to quit): 4
Number: 4.000000
Enter with a number ('q' to quit): 5
Number: 5.000000
Enter with a number ('q' to quit): 9
Number: 9.000000
Enter with a number ('q' to quit): -8
Number: -8.000000
Enter with a number ('q' to quit): +
Sum: 1.000000
Enter with a number ('q' to quit): +
Sum: 6.000000
Enter with a number ('q' to quit): +
Sum: 10.000000
Enter with a number ('q' to quit): +
Sum: 13.000000
Enter with a number ('q' to quit): q
pos 0> 13.000000
Press any key to continue . . . _
```

How to ensure robustness in the operation of the calculator?

Checking for empty stack and removing the used memory

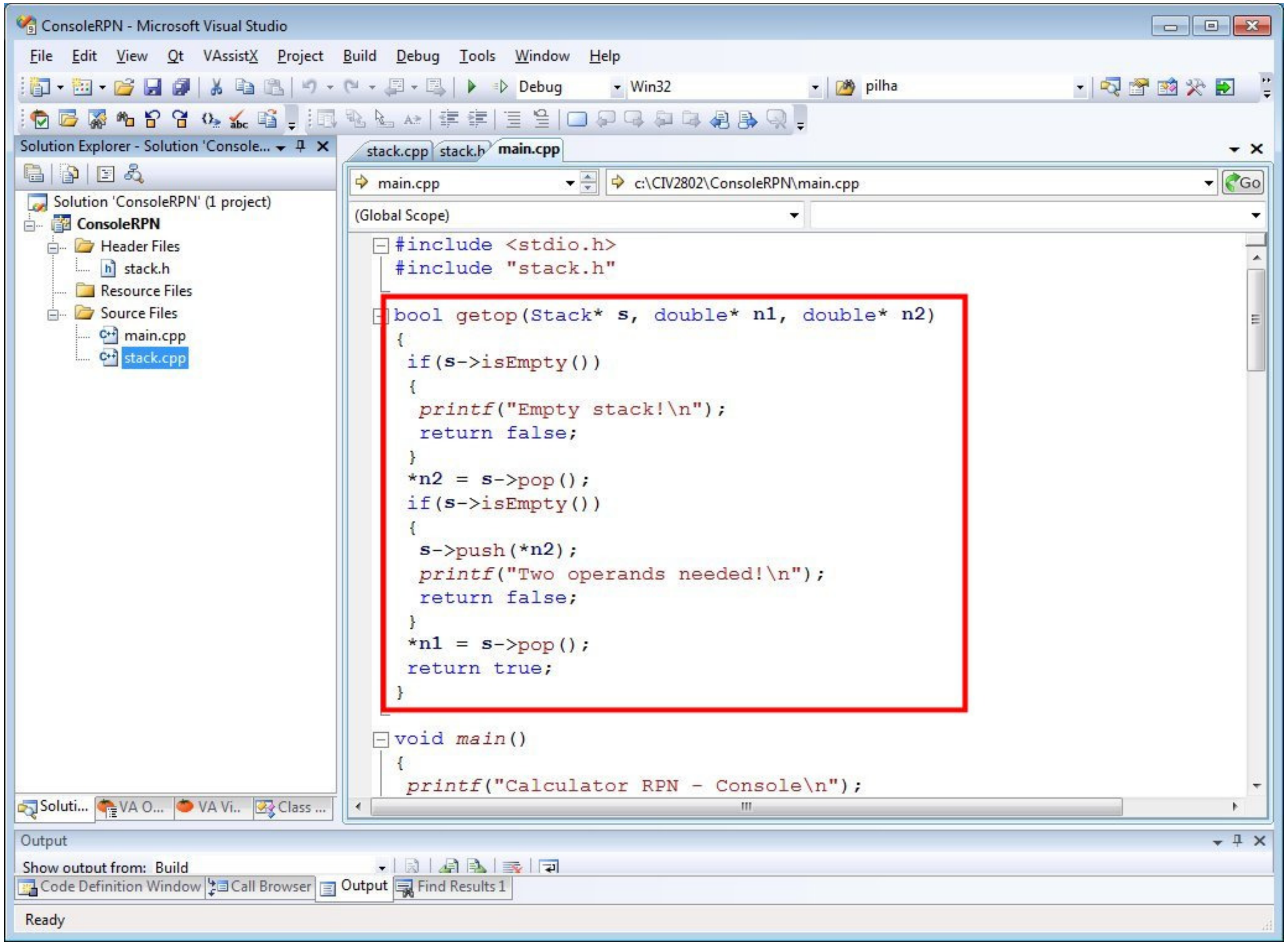

ConsoleRPN - Microsoft Visual Studio
File Edit View Qt VAssistX Project Build Debug Tools Window Help
Debug
Win32
pilha
Solution Explorer - Solution 'Console...
Solution 'ConsoleRPN' (1 project)
ConsoleRPN
Header Files
stack.h
Resource Files
Source Files
main.cpp
stack.cpp
stack.cpp
stack.h
main.cpp
c:\CIV2802\ConsoleRPN\stack.h
Go
(Global Scope)
Soluti...
VA O...
VA Vi...
Class ...
Output
Show output from:
Code Definition Window
Call Browser
Output
Find Results 1
Ready
ConsoleRPN - Microsoft Visual Studio
c:\CIV2802\ConsoleRPN\stack.cpp

```
#ifndef STACK_H
#define STACK_H

class Stack
{
public:
    Stack();
    ~Stack();
    void push( double n );
    double pop();
    bool isEmpty();
    void show();

private:
    int top;
    double *elems;
};

#endif
```

```
Stack::~Stack()
{
    delete elems;
    top = 0;
}

void Stack::push( double n )
{
    elems[top++]=n;
}

double Stack::pop()
{
    return elems[--top];
}

bool Stack::isEmpty()
{
    return top == 0;
}
```

What kind of problems might occur in the implementation and how to avoid them?

Auxiliary function to obtain both operands of an operation with the handling of possible errors



```
#include <stdio.h>
#include "stack.h"

bool getop(Stack* s, double* n1, double* n2)
{
 if(s->isEmpty())
 {
  printf("Empty stack!\n");
  return false;
 }
 *n2 = s->pop();
 if(s->isEmpty())
 {
  s->push(*n2);
  printf("Two operands needed!\n");
  return false;
 }
 *n1 = s->pop();
 return true;
}

void main()
{
 printf("Calculator RPN - Console\n");
```

What are the mechanisms for passing parameters to function in C/C++ language?

Mechanisms for passing parameters to function in C/C++

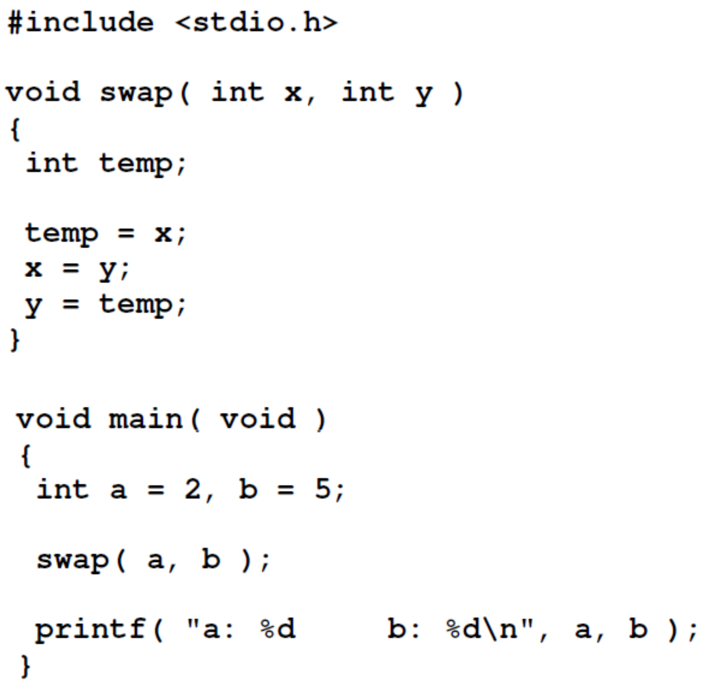
Consider the following program with a "swap" function whose purpose is to exchange values of two integers.

```
#include <stdio.h>

void swap( int x, int y )
{
 int temp;

 temp = x;
 x = y;
 y = temp;
}

 void main( void )
 {
  int a = 2, b = 5;

  swap( a, b );

  printf( "a: %d     b: %d\n", a, b );
 }
```

Write the result of the program, i.e. what is printed by the program?
Please, justify your answer.

Mechanisms for passing parameters to function in C/C++

Consider the following program with a "swap" function whose purpose is to exchange values of two integers.

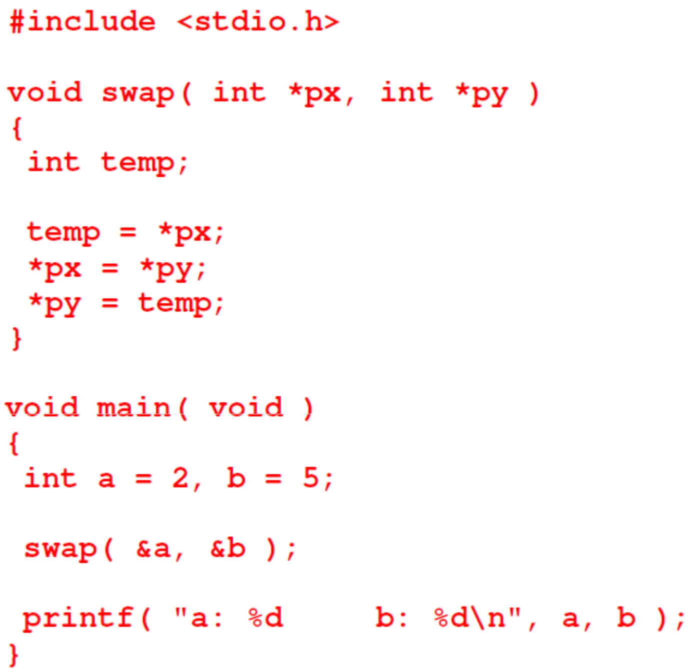


```
#include <stdio.h>

void swap( int x, int y )
{
 int temp;

 temp = x;
 x = y;
 y = temp;
}

 void main( void )
 {
  int a = 2, b = 5;

  swap( a, b );

  printf( "a: %d     b: %d\n", a, b );
 }
```

Program output:

```
a: 2     b: 5
```

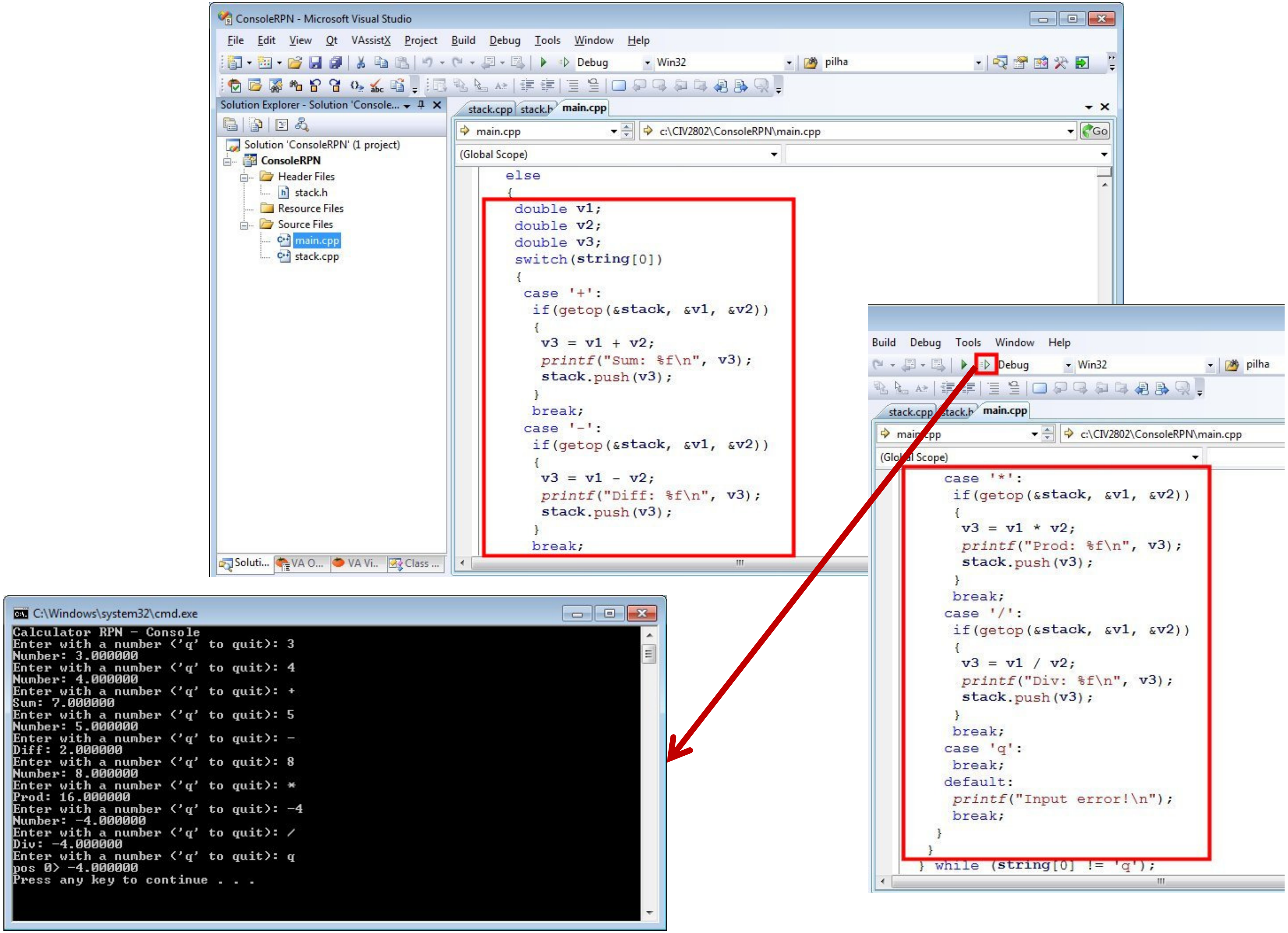
Corrected Program:



```
#include <stdio.h>

void swap( int *px, int *py )
{
 int temp;

 temp = *px;
 *px = *py;
 *py = temp;
}

void main( void )
{
 int a = 2, b = 5;

 swap( &a, &b );

 printf( "a: %d     b: %d\n", a, b );
}
```

It is observed that there was no exchange in the values of two numbers.
The reason is that the only mechanism for parameter passing to function in C/C++ is by value.
In this mechanism, a copy of the variable value is passed for the parameter.
The "swap" function is only changing the values of the copies, not the values of the variables "a" and "b".
The solution is to simulate a parameter passing by reference. This is done by passing (by value) the address of variables. The parameters of the "swap" function shall be pointers to integers.

Implementation of other operations on the calculator via the console



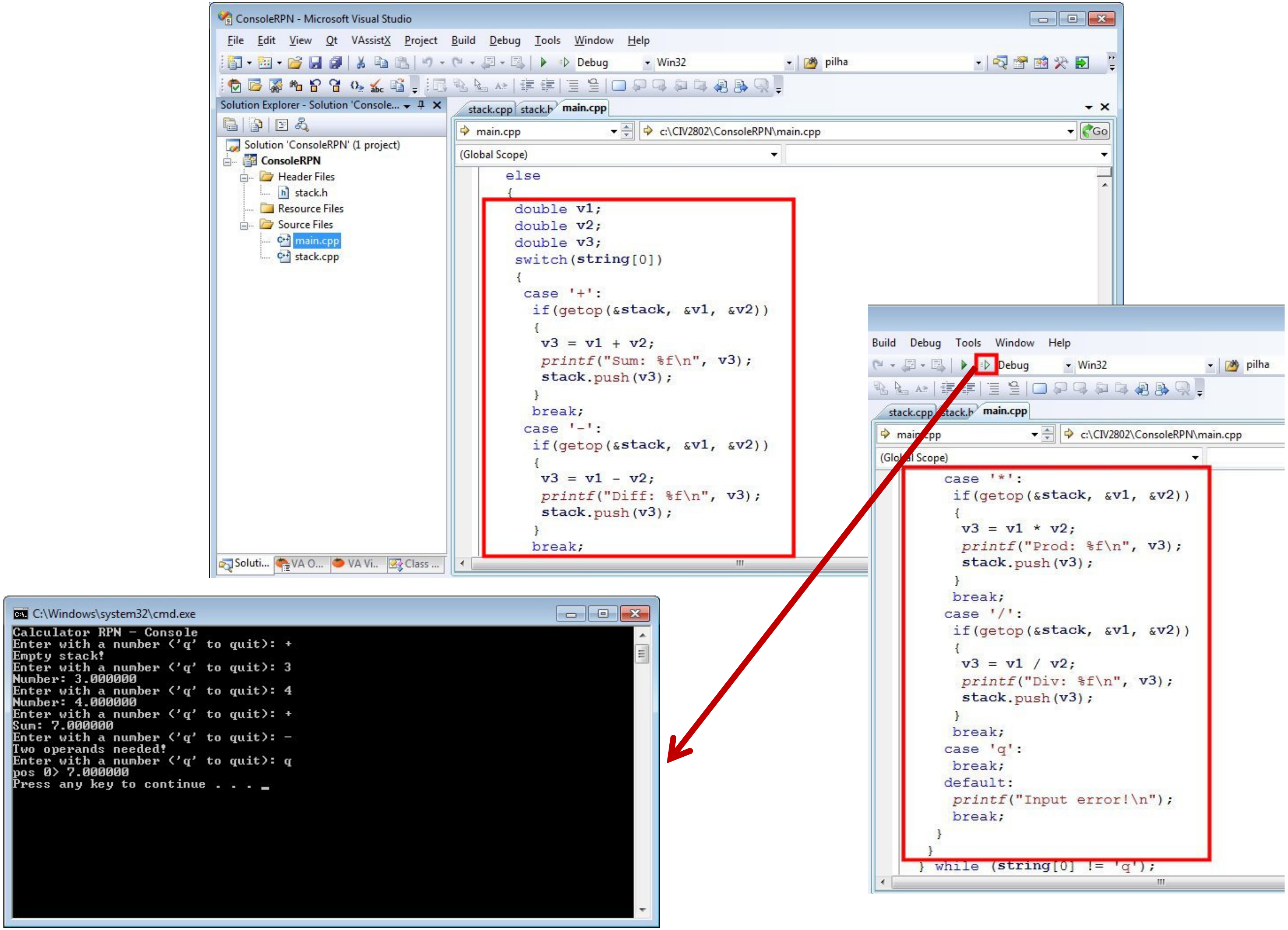
ConsoleRPN - Microsoft Visual Studio
File Edit View Qt VAssistX Project Build Debug Tools Window Help
Debug
Win32
pilha
Solution Explorer - Solution 'Console...
Solution 'ConsoleRPN' (1 project)
ConsoleRPN
Header Files
stack.h
Resource Files
Source Files
main.cpp
stack.cpp
stack.cpp
stack.h
main.cpp
c:\CIV2802\ConsoleRPN\main.cpp
Go
(Global Scope)
Soluti...
VA O...
VA Vi...
Class ...
C:\Windows\system32\cmd.exe

```
    else
    {
     double v1;
     double v2;
     double v3;
     switch(string[0])
     {
      case '+':
       if(getop(&stack, &v1, &v2))
       {
        v3 = v1 + v2;
        printf("Sum: %f\n", v3);
        stack.push(v3);
       }
       break;
      case '-':
       if(getop(&stack, &v1, &v2))
       {
        v3 = v1 - v2;
        printf("Diff: %f\n", v3);
        stack.push(v3);
       }
       break;
```

```
      case '*':
       if(getop(&stack, &v1, &v2))
       {
        v3 = v1 * v2;
        printf("Prod: %f\n", v3);
        stack.push(v3);
       }
       break;
      case '/':
       if(getop(&stack, &v1, &v2))
       {
        v3 = v1 / v2;
        printf("Div: %f\n", v3);
        stack.push(v3);
       }
       break;
      case 'q':
       break;
      default:
       printf("Input error!\n");
       break;
     }
    }
  } while (string[0] != 'q');
```

```
Calculator RPN - Console
Enter with a number ('q' to quit): 3
Number: 3.000000
Enter with a number ('q' to quit): 4
Number: 4.000000
Enter with a number ('q' to quit): +
Sum: 7.000000
Enter with a number ('q' to quit): 5
Number: 5.000000
Enter with a number ('q' to quit): -
Diff: 2.000000
Enter with a number ('q' to quit): 8
Number: 8.000000
Enter with a number ('q' to quit): *
Prod: 16.000000
Enter with a number ('q' to quit): -4
Number: -4.000000
Enter with a number ('q' to quit): /
Div: -4.000000
Enter with a number ('q' to quit): q
pos 0> -4.000000
Press any key to continue . . .
```

Testing the treatment of errors in the implementation of the program



ConsoleRPN - Microsoft Visual Studio
File Edit View Qt VAssistX Project Build Debug Tools Window Help
Debug
Win32
pilha
Solution Explorer - Solution 'Console...
Solution 'ConsoleRPN' (1 project)
ConsoleRPN
Header Files
stack.h
Resource Files
Source Files
main.cpp
stack.cpp
stack.cpp
stack.h
main.cpp
c:\CIV2802\ConsoleRPN\main.cpp
Go
(Global Scope)
Soluti...
VA O...
VA Vi...
Class ...
C:\Windows\system32\cmd.exe

```
    else
    {
     double v1;
     double v2;
     double v3;
     switch(string[0])
     {
      case '+':
       if(getop(&stack, &v1, &v2))
       {
        v3 = v1 + v2;
        printf("Sum: %f\n", v3);
        stack.push(v3);
       }
       break;
      case '-':
       if(getop(&stack, &v1, &v2))
       {
        v3 = v1 - v2;
        printf("Diff: %f\n", v3);
        stack.push(v3);
       }
       break;
```

```
      case '*':
       if(getop(&stack, &v1, &v2))
       {
        v3 = v1 * v2;
        printf("Prod: %f\n", v3);
        stack.push(v3);
       }
       break;
      case '/':
       if(getop(&stack, &v1, &v2))
       {
        v3 = v1 / v2;
        printf("Div: %f\n", v3);
        stack.push(v3);
       }
       break;
      case 'q':
       break;
      default:
       printf("Input error!\n");
       break;
     }
    }
   } while (string[0] != 'q');
```

```
Calculator RPN - Console
Enter with a number ('q' to quit): +
Empty stack!
Enter with a number ('q' to quit): 3
Number: 3.000000
Enter with a number ('q' to quit): 4
Number: 4.000000
Enter with a number ('q' to quit): +
Sum: 7.000000
Enter with a number ('q' to quit): -
Two operands needed!
Enter with a number ('q' to quit): q
pos 0> 7.000000
Press any key to continue . . . _
```