

Programa “ConsoleRPN”

Programa criado com o Visual Studio 2008
para efetuar operações algébricas entre números,
uma calculadora funcionando com console usando RPN

Objetivos da Aula:

Apresentar os conceitos de programação da linguagem C/C++ utilizando um exemplo de uma calculadora RPN (Reversed Polish Notation)

Conteúdo/Assuntos Abordados:

Introdução à linguagem de programação C/C++.

Introdução ao conceito de classes e objetos em C++.

Competências/Habilidades:

Conhecer os recursos básicos da linguagem C/C++.

Capacidade de criar um projeto console no Visual Studio 2008.

Capacidade de entendimento de algoritmos e estruturas de dados.

Entendimento do uso de funções e os mecanismos de passagem de parâmetros para funções na linguagem C/C++.

O aluno deverá ser capaz de desenvolver aplicativos práticos.

O que significa uma calculadora RPN (Reserved Polish Notation)?

W Notação polonesa inversa x

pt.wikipedia.org/wiki/Notação_polonesa_inversa

Aplicativos Gmail BOL LSC Mestrado CompGraph Ensino Pesquisa Petroleo Outros Edital - PET PERM ASCE USGS Limit Analysis and S... Outros favoritos

Criar conta Entrar

Artigo Discussão Ler Editar Editar código-fonte Ver histórico Pesquisa

WIKIPÉDIA
A enciclopédia livre

Página principal
Conteúdo destacado
Eventos atuais
Esplanada
Página aleatória
Portais
Informar um erro

Colaboração
Boas-vindas
Ajuda
Página de testes
Portal comunitário
Mudanças recentes
Manutenção
Criar página
Páginas novas
Contato
Donativos

Imprimir/exportar

Ferramentas

Noutras línguas

Беларуская
Català
Čeština
Dansk
Deutsch
English
Esperanto
Español
Euskara
فارسی
Suomi
Français

Modificação dos nossos Termos de Uso:
Por favor, comente sobre uma proposta de alteração relativa a edições pagas não reveladas.
[Ajuda-nos com as traduções!]

Notação polonesa inversa

Origem: Wikipédia, a enciclopédia livre.

Notação Polonesa Inversa (ou **RPN** na sigla em inglês, de *Reverse Polish Notation*), também conhecida como **notação pós-fixada**, foi inventada pelo filósofo e cientista da computação australiano **Charles Hamblin** em meados dos anos 1950, para habilitar armazenamento de memória de endereço zero. Ela deriva da **notação polonesa**, introduzida em 1920 pelo matemático polonês Jan Łukasiewicz. (Daí o nome sugerido de *notação Zciweisakul*.) Hamblin apresentou seu trabalho numa conferência em Junho de 1957, e o publicou em 1957 e 1962. Conquanto rejeitado em primeira apreciação por parte da maioria dos utilizadores, sob a alegação de ser "muito difícil, preferindo-se a convencional", tudo não passa de apenas impressão primeira de quem não tem familiaridade com a nova notação e, pois, com as suas vantagens. Quer na computação automatizada, quer no cálculo manual assistido por instrumentos de cálculo (*calculadoras, lato sensu*), a notação polonesa reversa (RPN) apresenta as seguintes vantagens:

1. Reduz o número de passos lógicos para se perfazerem operações binárias e, posto que as demais operações são ou binárias puras compostas, ou binárias compostas com unitárias ou apenas unitárias, o número total de passos lógicos necessários a um determinado cômputo será sempre menor que aquele que utiliza a sintaxe convencional (lógica algébrica direta);
2. Trabalha com **pares ordenados a priori**, somente definindo a lei de composição binária aplicável após a eleição e a introdução do desejado par no cenário de cálculo. Até o momento final, se poderá decidir pela troca ou pela permanência da operação original.
3. Minimiza os erros de computação, automática ou manual assistida;
4. Maximiza a velocidade operacional na solução de problemas.

Tudo isso pode ser facilmente constatado na tabela a seguir, por meio de contagem de números de passos lógicos operacionais para o modo RPN comparado com o modo convencional. A notação RPN tem larga utilização no mundo científico pela fama de permitir uma linha de raciocínio mais direta durante a formulação e por dispensar o uso de parênteses mas mesmo assim manter a ordem de resolução.

ALGUNS EXEMPLOS DE OPERAÇÕES E NOTAÇÕES

Operação	Notação convencional	Notação Polonesa	Notação Polonesa Inversa
$a + b$	a+b	+ a b	a b +
$\frac{a + b}{c}$	(a+b)/c	/ + a b c	a b + c /
$\frac{a \cdot b - c \cdot d}{e \cdot f}$	((a*b)-(c*d))/(e*f)	/ - * a b * c d * e f	a b * c d * - e f * /

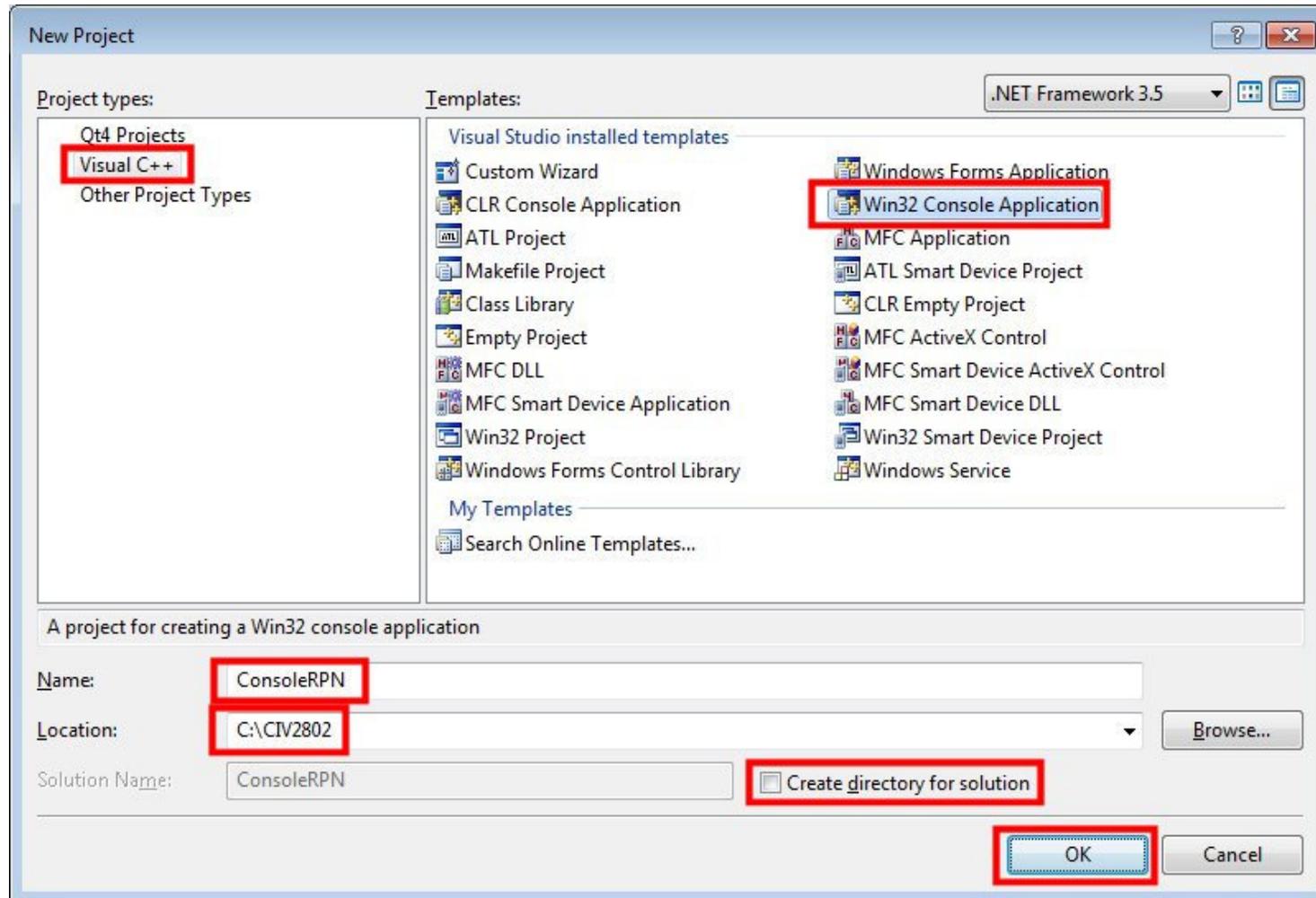
Exemplo clássico de calculadora que utiliza a Notação Polonesa Reversa (RPN)



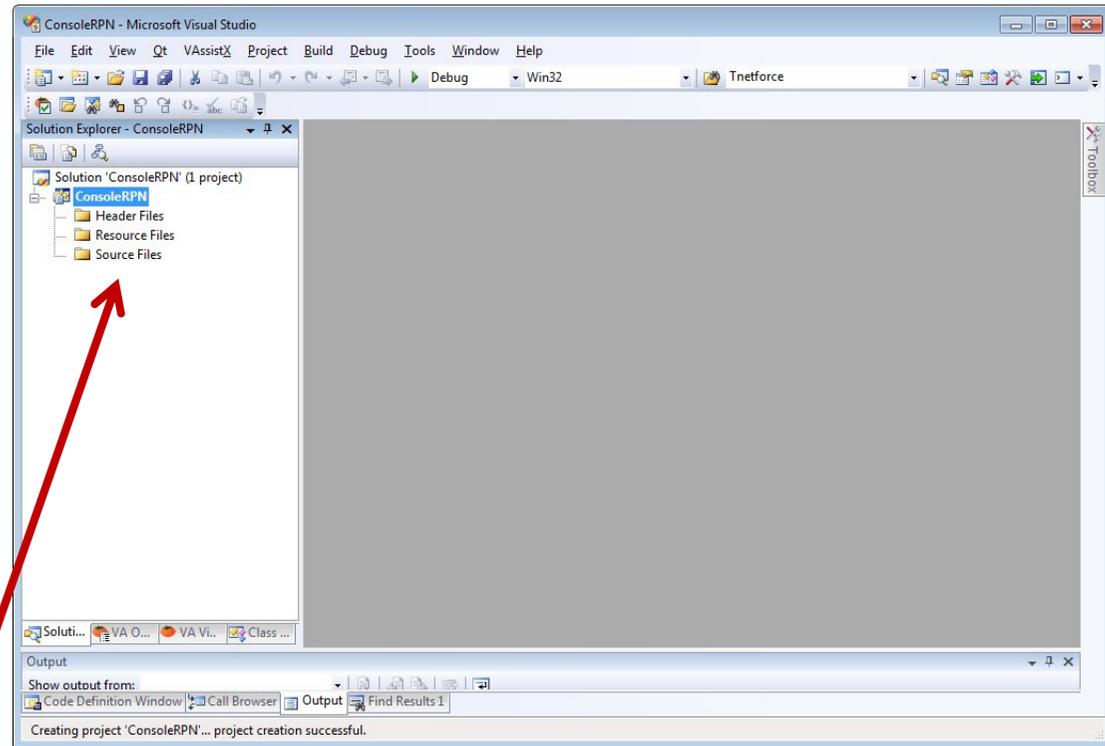
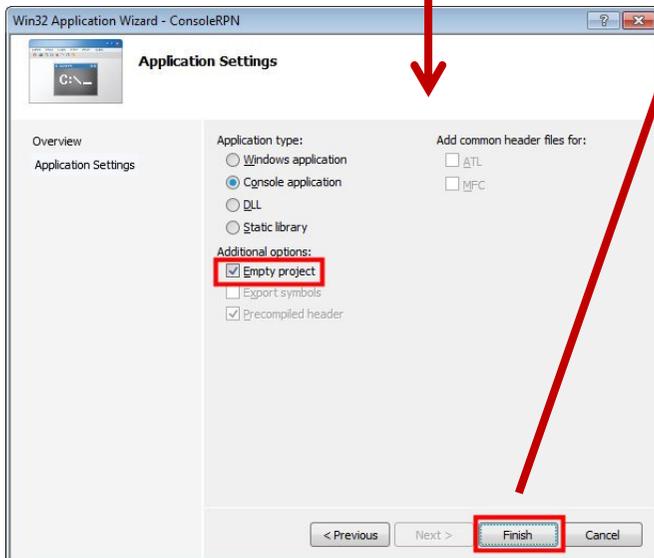
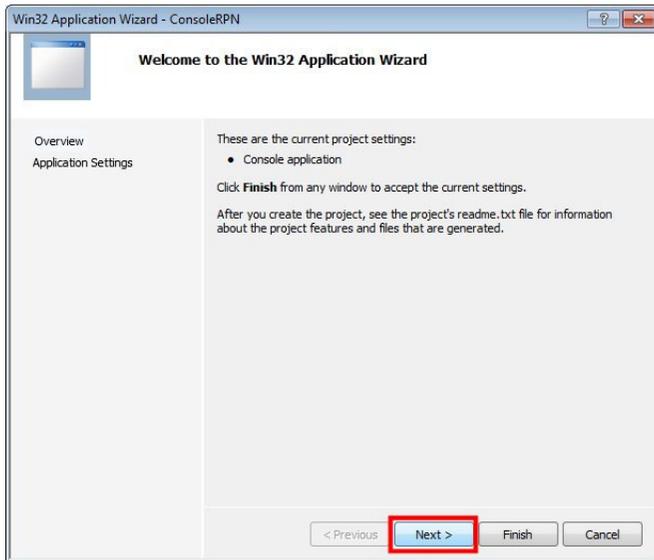
Como criar nossa própria calculadora utilizando uma linguagem de programação e uma interface gráfica?

Criação de um novo projeto no Visual Studio 2008 do tipo Console

Vamos começar programando uma calculadora sem interface gráfica.

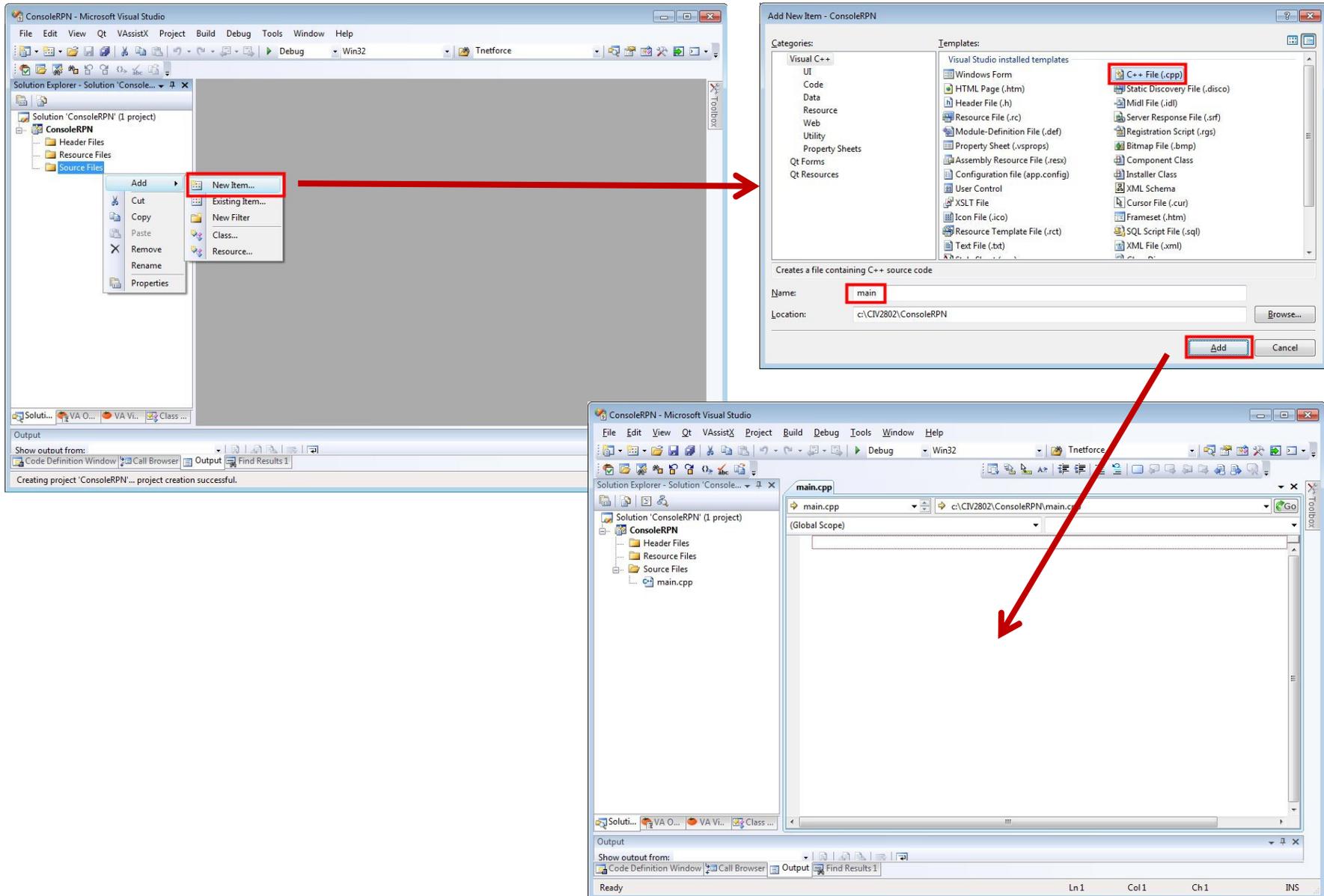


Wizard para criação de um projeto do tipo Console “Vazio”



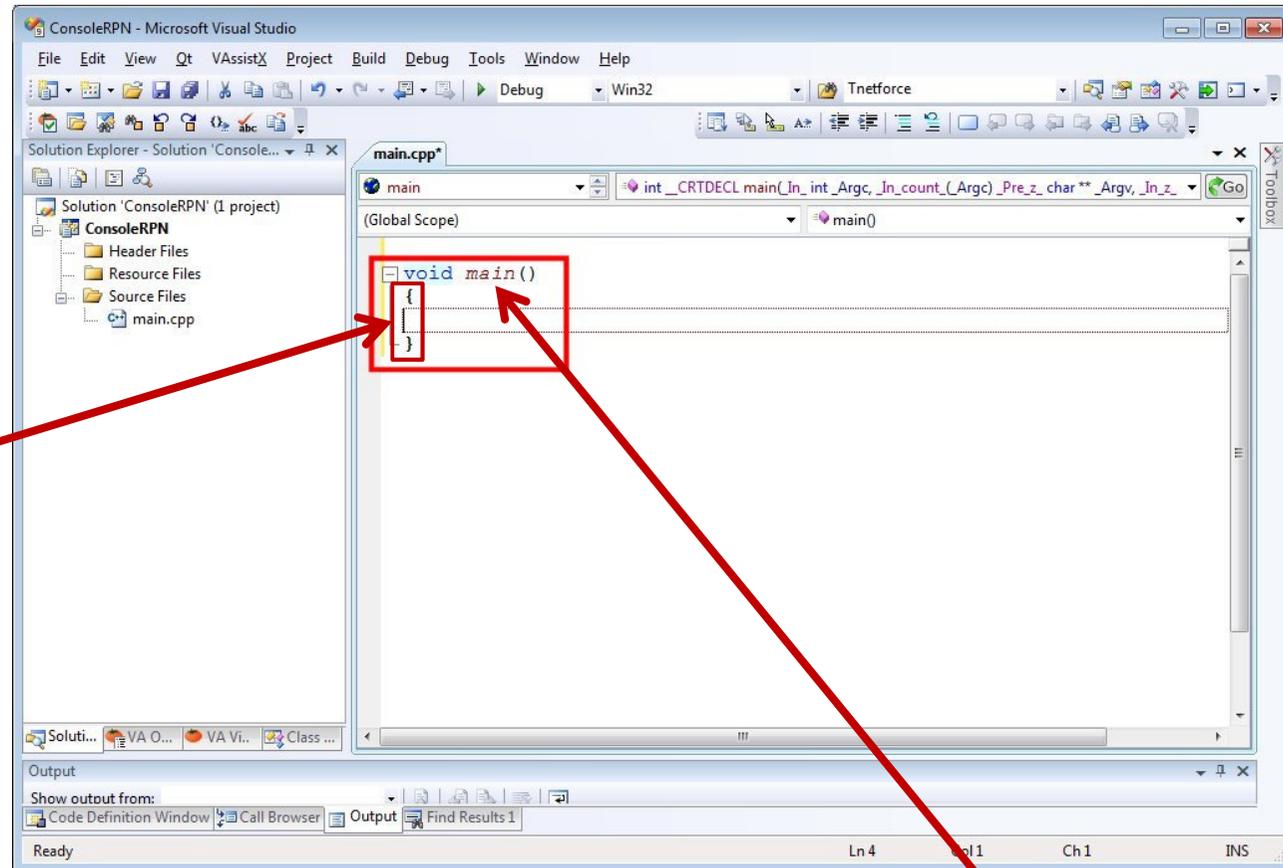
Como começar desenvolver um programa do zero?

Criação de um arquivo novo, com o nome “main” e extensão “.cpp”



Implementação do arquivo “main.cpp”

Estrutura básica de um programa em C



Um par de chaves define um bloco de código

A execução do programa inicia pela chamada da função **main**.

A função main pode ter diferentes assinaturas:

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

Todo programa em C contém pelo menos uma função:

main

Nesse caso `void main()` não requer parâmetros e não retorna parâmetros.

Documentação e referências para programar em C e C++

Como imprimir dados na janela de um programa console?

www.cplusplus.com

Information
Tutorials
Reference
Articles
Forum

Welcome to **cplusplus.com**

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 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 in categories:

- All Admins
- 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
- Business

C++ Search
Search this website:

Other tools are also available to search this website:

www.cplusplus.com/reference/clibrary/

Reference C library

library

C 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.h>`.
- 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++.

There 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`, `strchr`, `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 `<uchar>` (as by ISO/IEC 19769:2004).

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

Headers

Reference

- `<cassert>` (`assert.h`)
- `<ctype>` (`ctype.h`)
- `<cerrno>` (`errno.h`)
- `<cfenv>` (`fenv.h`)
- `<float>` (`float.h`)
- `<inttypes>` (`inttypes.h`)
- `<iso646>` (`iso646.h`)
- `<climits>` (`limits.h`)
- `<locale>` (`locale.h`)
- `<cmath>` (`math.h`)
- `<csignal>` (`signal.h`)
- `<stdarg>` (`stdarg.h`)
- `<stdbool>` (`stdbool.h`)
- `<stddef>` (`stddef.h`)
- `<stdint>` (`stdint.h`)
- `<stdio>` (`stdio.h`)
- `<stdlib>` (`stdlib.h`)
- `<string>` (`string.h`)
- `<tgmath>` (`tgmath.h`)
- `<ctime>` (`time.h`)
- `<uchar>` (`uchar.h`)
- `<wchar>` (`wchar.h`)
- `<cwctype>` (`wctype.h`)

Containers:
Input/Output:

www.cplusplus.com/reference/cstdio/

header

<stdio.h> (stdio.h)

C library to perform Input/Output operations

Input and Output operations can also be performed in C++ using the C Standard Input and Output Library (`<stdio.h>`, known as `stdio.h` in the C language). This library provides 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 a 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 `<stdio.h>` 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 in 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

Imprimindo mensagem no console usando a biblioteca C padrão I/O

The image shows two screenshots of Microsoft Visual Studio. The top screenshot displays the source code in `main.cpp` with the following content:

```
#include <stdio.h>

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

The bottom screenshot shows the `Build` menu with `Build ConsoleRPN` selected, and the `Output` window displaying the build process:

```
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:\GIV2802\ConsoleRPN\Debug\BuildLog.htm"
1>ConsoleRPN - 0 error(s), 0 warning(s)
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

Red boxes and arrows highlight key elements: `#include <stdio.h>`, `printf("Calculator RPN - Console\n");`, and the `Build ConsoleRPN` menu item.

Informa ao compilador que as funções de I/O padrão serão usadas.

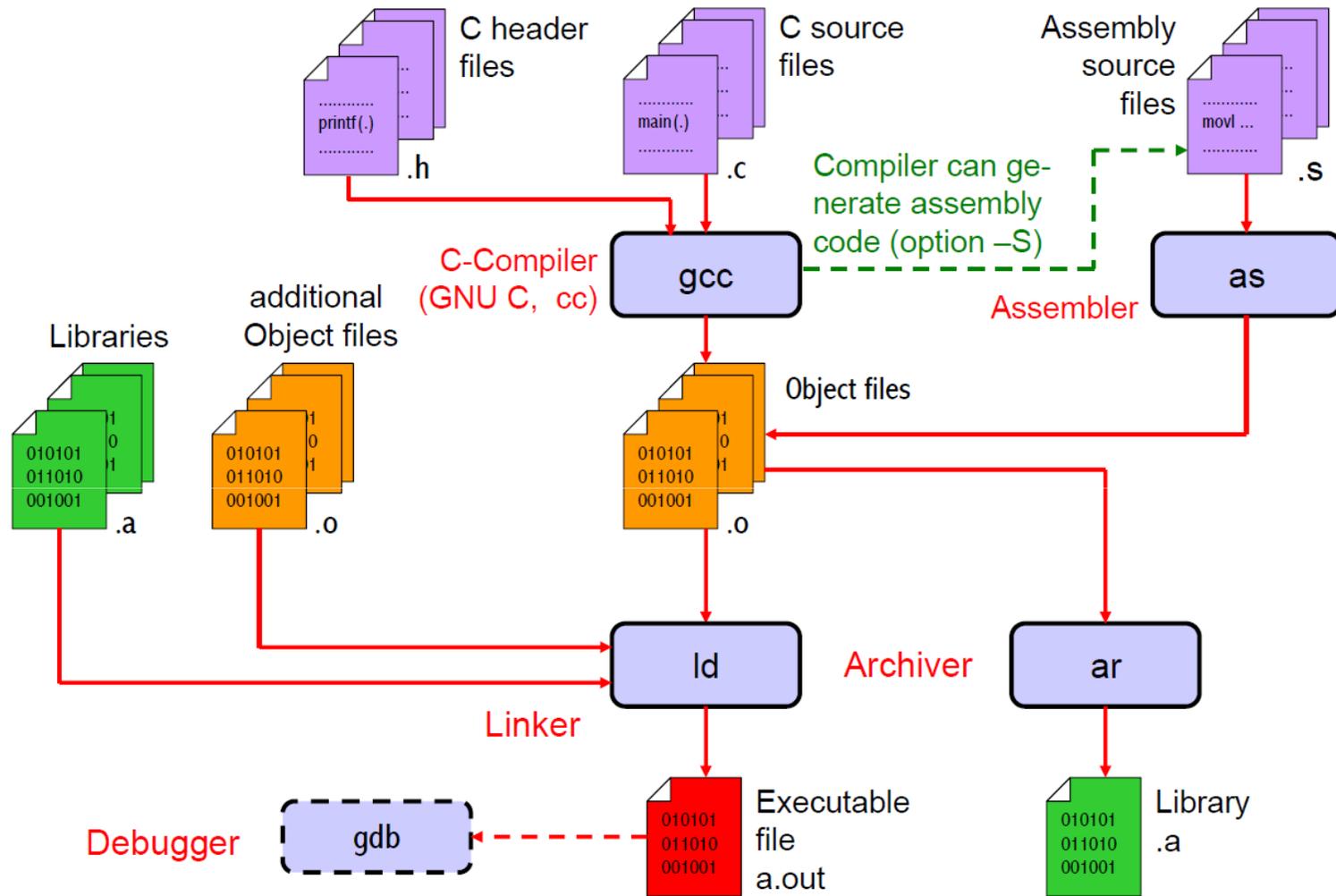
\n produz uma nova linha no console.

Escreve uma *string* na saída padrão do console. `printf` é uma função de C que é parte da biblioteca C padrão.

Toda declaração termina com um ponto e vírgula, exceto para a chave que fecha um bloco (`}`).

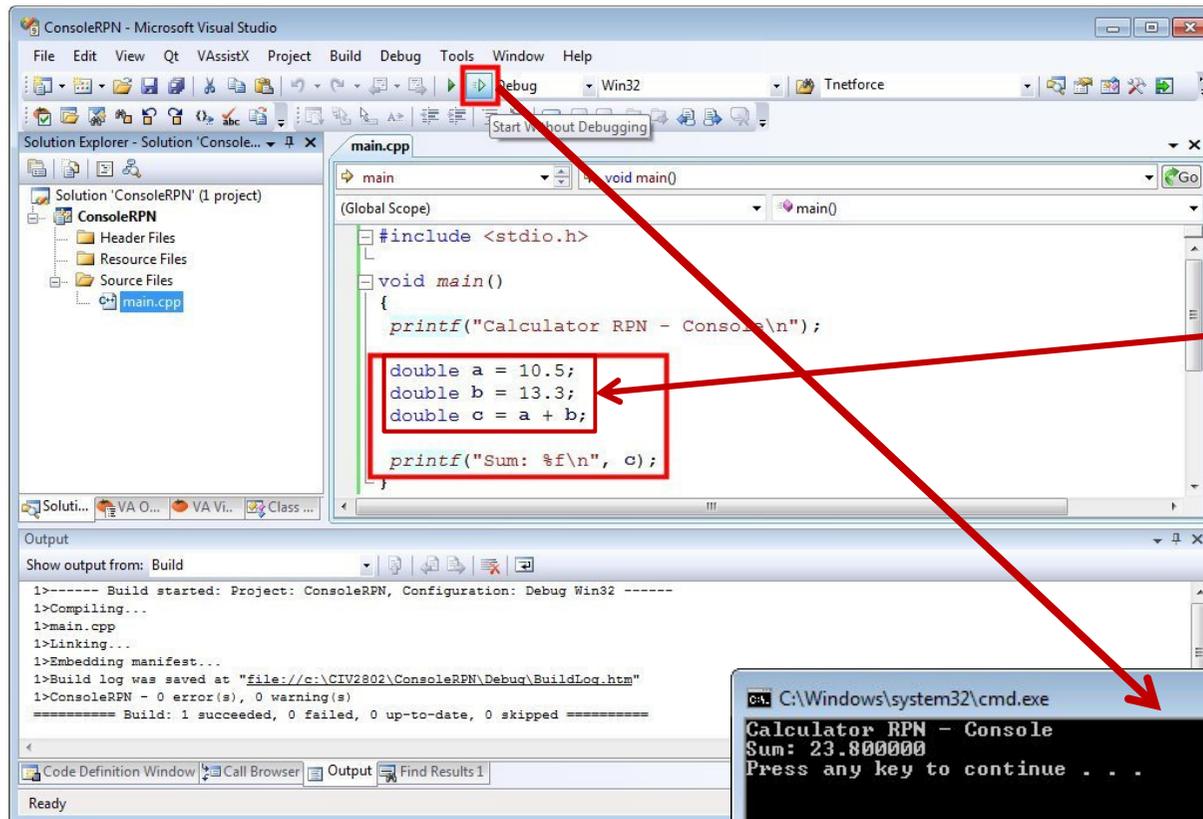
O que acontece quando o programa é construído (compilado e linkado)?

De códigos em C para um binário executável

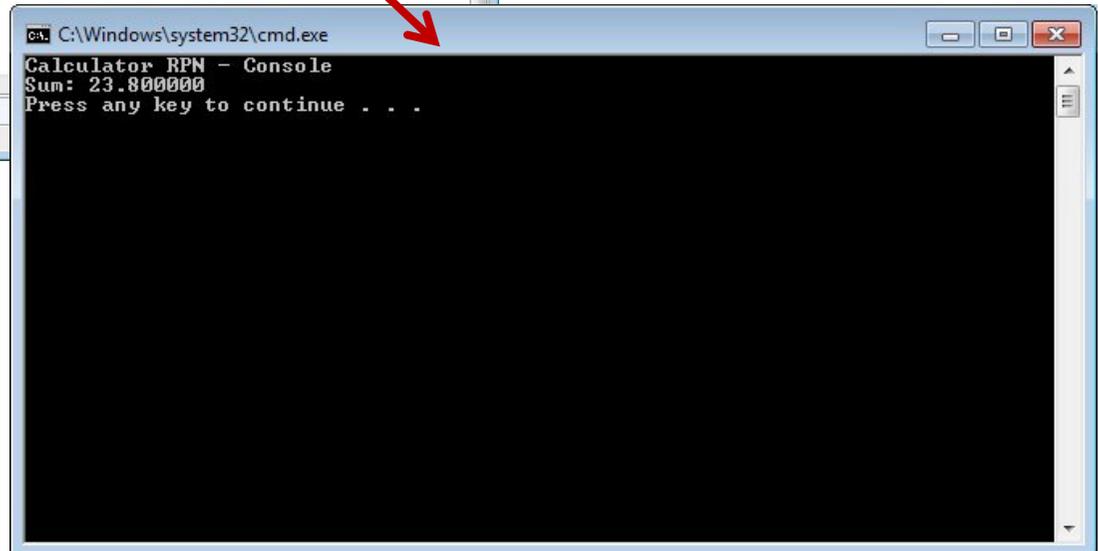


Fonte: René Müller, Introduction to the C-Language and Programming Environment, Winter Semester 2005/06

Imprimindo no console (saída) o resultado da soma de dois números



Variáveis locais, apenas disponíveis no escopo de `main()`



Quais são os tipos de variáveis nativas e operadores da linguagem C/C++?

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

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

C operators

Type	Operator	Action
Arithmetic	-	Subtraction
	+	Addition
	*	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 shift
Miscellaneous	?	Ternary (y=x>9?100:200)
	& and *	Pointer operators
	sizeof	Width of a datatype (in bytes)
	. and ->	Access to structures
	[]	Access to arrays

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

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

Variable declaration and scope

- Variables **MUST** 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 or 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 */
```

Função (*function*)? O que é isso?

Como fazer a operação de soma utilizando uma função?

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

Implementando uma função para executar a soma de dois números

The screenshot shows the Visual Studio IDE with a C++ project named 'ConsoleRPN'. The main.cpp file contains the following code:

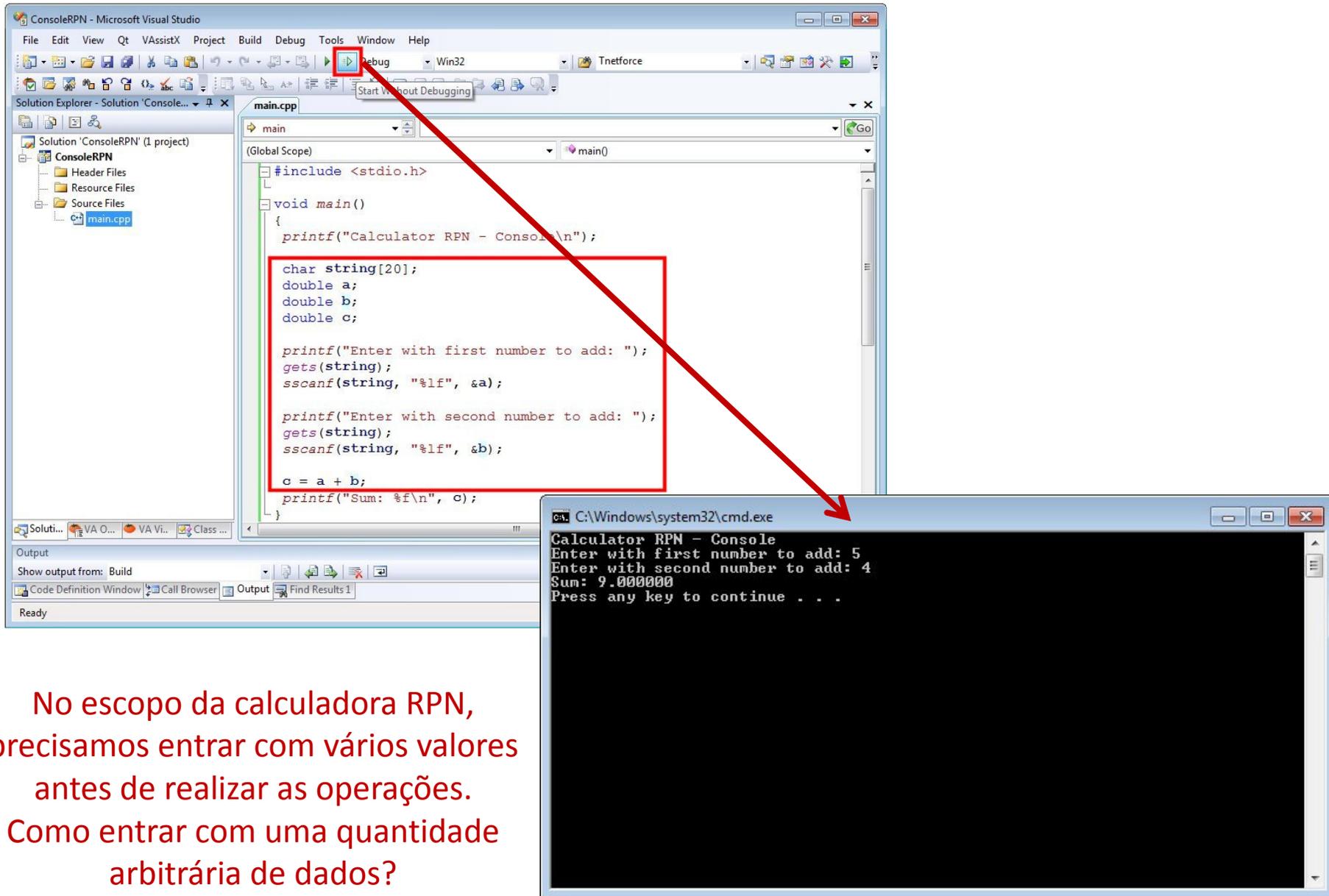
```
1 #include <stdio.h>
2
3 double sum(double n1, double n2);
4
5 void main()
6 {
7     printf("Calculator RPN -- Console\n");
8
9     double a = 10.5;
10    double b = 13.3;
11    double c = sum(a, b);
12
13    printf("Sum: %f\n", c);
14 }
15
16 /* Function to add two double variables.
17  * Returns as function value the result.
18  */
19 double sum(double n1, double n2)
20 {
21     return n1 + n2;
22 }
```

Annotations in red boxes with arrows pointing to the code:

- Declaração (prototype) da função sum**: Points to line 3: `double sum(double n1, double n2);`
- Chamada da função sum**: Points to line 11: `double c = sum(a, b);`
- Um par de `/* */` define um comentário que é ignorado pelo compilador**: Points to lines 16-18: `/* Function to add two double variables. * Returns as function value the result. */`
- Implementação da função sum**: Points to lines 19-22: `double sum(double n1, double n2) { return n1 + n2; }`
- A função sum recebe dois valores double na lista de parâmetros**: Points to the parameters `double n1, double n2` in the function definition on line 19.
- A função sum retorna um valor double**: Points to the return type `double` on line 19.

Como realizar entrada de dados na linguagem C/C++?

Inserindo pelo console (entrada) os dois números a serem somados



The image shows a screenshot of Microsoft Visual Studio with a C++ project named 'ConsoleRPN'. The code in 'main.cpp' is as follows:

```
#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);
}
```

A red box highlights the input handling section of the code. A red arrow points from this box to a console window titled 'C:\Windows\system32\cmd.exe'. The console output is:

```
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 . . .
```

No escopo da calculadora RPN, precisamos entrar com vários valores antes de realizar as operações. Como entrar com uma quantidade arbitrária de dados?

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;  
}
```

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

Preparando o programa para inserir uma quantidade qualquer de números, e criação de comando para encerrar o programa

```
#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');
}
```

```
C:\Windows\system32\cmd.exe
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 . . .
```

Qual a melhor estratégia para organizar, guardar e acessar os dados inseridos?

Estruturas de Dados

pt.wikipedia.org/wiki/Estrutura_de_dados

Wikipédia A enciclopédia livre

Artigo Discussão Ler Editar Editar código-fonte Ver histórico Pesquisa

Estrutura de dados

Origem: Wikipédia, a enciclopédia livre.

Na **Ciência da computação**, uma **estrutura de dados** é um modo particular de armazenamento e organização de dados em um computador de modo que possam ser usados eficientemente.^{1 2}

Diferentes tipos de estrutura de dados são adequadas a diferentes tipos de aplicação e algumas são altamente especializadas, destinando-se a algumas tarefas específicas. Por exemplo, as **B-trees** são particularmente indicadas para a implementação de **bases de dados**, enquanto que a implementação de **compiladores** geralmente requer o uso de **tabela de dispersão** para a busca de identificadores.

Estruturas de dados e **algoritmos** são temas fundamentais da **ciência da computação**, sendo utilizados nas mais diversas áreas do conhecimento e com os mais diferentes propósitos de aplicação. Sabe-se que algoritmos manipulam dados. Quando estes dados estão organizados (dispostos) de forma coerente, caracterizam uma forma, uma **estrutura de dados**. A organização e os métodos para manipular essa estrutura é que lhe conferem singularidade e diminuição do espaço ocupado pela memória RAM, além de tornar o código-fonte do projeto simplificado.

As estruturas de dados são chamadas tipos de dados compostos que dividem-se em homogêneos (vetores (registros)). As estruturas homogêneas são conjuntos de dados formados pelo mesmo tipo de dado primitivo heterogêneas são conjuntos de dados formados por tipos de dados primitivos diferentes (campos do registro). A escolha de uma estrutura de dados apropriada pode tornar um problema complicado em um simples. O estudo das estruturas de dados está em constante desenvolvimento (assim como o de algoritmos) existem certas estruturas clássicas que se comportam como padrões.

Índice [esconder]

- 1 Estruturas de dados clássicas
 - 1.1 Vetores ou arrays
 - 1.2 Lista
 - 1.3 Fila
 - 1.4 Pilha
 - 1.5 Árvores
 - 1.5.1 Árvores binárias
 - 1.6 Grafo
 - 1.7 Deque
 - 1.8 Tabela de hashing
- 2 Referências
- 3 Ver também

Estruturas de dados clássicas

[editar | editar código-fonte]

Vetores ou arrays

[editar | editar código-fonte]

Ver artigo principal: *Array*

Vetores^{FE}, ou **vetores**^{FE} ou **arrays** são estruturas de dados lineares e estáticas, isto é, são compostas por um número fixo (finito) de elementos de um determinado tipo de dados. O tempo de acesso aos elementos de um vetor é muito rápido, sendo considerado constante: o acesso aos elementos é feito pelo seu índice no vetor. Porém, a remoção de elementos pode ser custosa se não for desejável que haja espaços "vazios" no meio do vetor, pois nesse caso é necessário "arrastar" de uma posição todos os elementos depois do elemento removido.

Essa é uma estrutura muito recomendada para casos em que os dados armazenados não mudarão, ou pouco mudarão, através do tempo.

Lista

[editar | editar código-fonte]

Ver artigo principal: *Lista*

Uma Lista é uma estrutura de dados linear. Uma **lista ligada**, também chamada de encadeada, é linear e dinâmica, é composta por nós que apontam para o próximo elemento da lista, o último elemento apontará para nulo. Para compor uma lista encadeada, basta guardar seu primeiro elemento.

Fila

[editar | editar código-fonte]

Ver artigo principal: *FIFO*

As filas são estruturas baseadas no princípio FIFO (*first in, first out*), em que os elementos que foram inseridos no início são os primeiros a serem removidos. Uma fila possui duas funções básicas: **ENQUEUE**, que adiciona um elemento ao final da fila, e **DEQUEUE**, que remove o elemento no início da fila. A operação **DEQUEUE** só pode ser aplicada se a fila não estiver vazia, causando um erro de **underflow** ou fila vazia se esta operação for realizada nesta situação.

Pilha

[editar | editar código-fonte]

Ver artigo principal: *LIFO*

A pilha é uma estrutura de dados baseada no princípio LIFO (*last in, first out*), na qual os dados que foram inseridos primeiros na pilha serão os últimos a serem removidos. Existem duas funções que se aplicam a todas as pilhas: **PUSH**, que insere um dado no topo da pilha, e **POP**, que remove o item no topo da pilha.

Uma árvore binária é uma estrutura de dados.

Qual a estrutura de dados mais adequada para o problema da calculadora RPN?

A Estrutura de Dados Pilha (*Stack*)

Artigo [Discussão](#) [Ler](#) [Editar](#) [Editar código-fonte](#) [Ver histórico](#)

Pilha (informática)

Origem: Wikipédia, a enciclopédia livre.

Em ciência da computação, uma **pilha** (**stack** em inglês) é um tipo abstrato de dado e estrutura de dados baseado no princípio de *Last In First Out* (LIFO). Pilhas são usadas extensivamente em cada nível de um sistema de computação moderno. Por exemplo, um PC moderno usa pilhas ao nível de arquitetura, as quais são usadas no design básico de um sistema operacional para manipular interrupções e chamadas de função do sistema operacional. Entre outros usos, pilhas são usadas para executar uma Máquina virtual java e a própria linguagem Java possui uma classe denominada "Stack", as quais podem ser usadas pelos programadores. A pilha é onipresente.

Um sistema informático baseado em pilha é aquele que armazena a informação temporária basicamente em pilhas, em vez de registradores de hardware da UCP (um sistema baseado em registradores).

Índice [\[esconder\]](#)

- 1 História
- 2 Ver também
- 3 Referências
- 4 Ligações externas

História

A pilha foi inicialmente proposta em 1955, e patenteada em 1957, pelo alemão Friedrich Ludwig Bauer.¹ O mesmo conceito foi desenvolvido, por volta da mesma época, pelo australiano Charles Leonard Hamblin.

Ver também

Como implementar uma estrutura de dados de pilha na linguagem C/C++?

Programação Orientada a Objetos

Firefox

W Orientação a objetos – Wikipédia, a enci... +

pt.wikipedia.org/wiki/Orientação_a_objetos

programação orientada a objetos

Criar uma conta Autenticação

Artigo Discussão

Ler Editar Editar código-fonte Ver histórico Pesquisa

Orientação a objetos

Origem: Wikipédia, a enciclopédia livre.

Este artigo ou se(c)ção cita uma ou mais fontes fiáveis e independentes, mas ela(s) não cobre(m) todo o texto.

Por favor, [melhore](#) este artigo providenciando mais [fontes fiáveis e independentes](#) e [inserindo-as](#) em [notas de rodapé](#) ou no corpo do texto, conforme o [livro de estilo](#).

Encontre fontes: [Google](#) — [notícias](#), [livros](#), [académico](#) — [Scirus](#) — [Bing](#). [Veja como referenciar e citar as fontes.](#)

A **orientação a objetos** é um **paradigma** de **análise**, **projeto** e **programação** de sistemas de *software* baseado na composição e interação entre diversas unidades de software chamadas de objetos.

Em alguns contextos, prefere-se usar **modelagem** orientada ao objeto, em vez de programação. De fato, o paradigma "orientação a objeto", tem bases conceituais e origem no campo de estudo da cognição, que influenciou a área de **inteligência artificial** e da **linguística**, no campo da abstração de conceitos do mundo real. Na qualidade de método de modelagem, é tida como a melhor estratégia para se eliminar o "gap semântico", dificuldade recorrente no processo de modelar o mundo real do domínio do problema em um conjunto de componentes de software que seja o mais fiel na sua representação deste domínio. Facilitaria a comunicação do profissional modelador e do usuário da área alvo, na medida em que a correlação da simbologia e conceitos abstratos do mundo real e da ferramenta de modelagem (conceitos, terminologia, símbolos, grafismo e estratégias) fosse a mais óbvia, natural e exata possível.

Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de *software*. Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos. C++, C#, VB.NET, Java, Object Pascal, Objective-C, Python, SuperCollider, Ruby e Smalltalk são exemplos de **linguagens de programação orientadas a objetos**. ActionScript, ColdFusion, Javascript, PHP (a partir da versão 4.0), Perl (a partir da versão 5) e Visual Basic (a partir da versão 4) são exemplos de **linguagens de programação** com suporte a orientação a objetos.

Índice [\[esconder\]](#)

- 1 [Conceitos essenciais](#)
- 2 [Referências](#)
- 3 [Bibliografia](#)
- 4 [Ver também](#)
- 5 [Ligações externas](#)

Orientação a objetos

- [Objeto / Instância](#)
- [Classe](#)
- [Abstração](#)
- [Métodos](#)
- [Atributo](#)
- [Encapsulamento](#)
- [Herança](#)
- [Herança múltipla](#)
- [Polimorfismo](#)
- Outras referências**
- [Padrões de projeto](#)
- [UML](#)
- [Engenharia OO](#)

Como implementar uma pilha usando orientação a objetos em C++?

Implementando uma Pilha (Stack): criação do arquivo "stack.h"

The image illustrates the steps to create a new header file in Visual Studio:

- Top Left:** The Visual Studio interface shows the 'Solution Explorer' with the 'Header Files' folder selected. A context menu is open, and 'New Item...' is highlighted. A red arrow points from this menu item to the 'Add New Item' dialog.
- Top Right:** The 'Add New Item - ConsoleRPN' dialog is shown. Under the 'Visual C++' category, 'Header File (.h)' is selected and highlighted with a red box. The 'Name' field contains 'stack.h' and the 'Location' is 'c:\CV2802\ConsoleRPN'. The 'Add' button is highlighted with a red box.
- Bottom:** The Visual Studio interface shows the newly created 'stack.h' file open in the editor. The code defines a 'Stack' class with public methods 'push', 'pop', and 'show', and a private member 'top' and 'elems'. The code is enclosed in a preprocessor guard. A red box highlights the entire code block, and a red arrow points from the 'Add' button in the dialog to this code block.

```
#ifndef STACK_H
#define STACK_H

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

private:
    int top;
    double *elems;
};

#endif
```

Implementando uma Pilha (Stack): criação do arquivo "stack.cpp"

The image illustrates the process of creating a C++ source file for a stack implementation in Visual Studio. It is divided into three main sections:

- Top-Left:** A screenshot of the Visual Studio IDE showing the 'Add' menu with 'New Item...' selected. A red arrow points from this menu item to the 'Add New Item' dialog.
- Top-Right:** The 'Add New Item - ConsoleRPN' dialog box. The 'C++ File (.cpp)' template is selected in the 'Templates' list. The 'Name' field is filled with 'stack.cpp' and the 'Location' is 'c:\CIV2802\ConsoleRPN'. The 'Add' button is highlighted with a red box.
- Bottom-Left:** A screenshot of the Visual Studio IDE showing the code editor with the implementation of the 'Stack' class in 'stack.cpp'. A red box highlights the code:

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

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

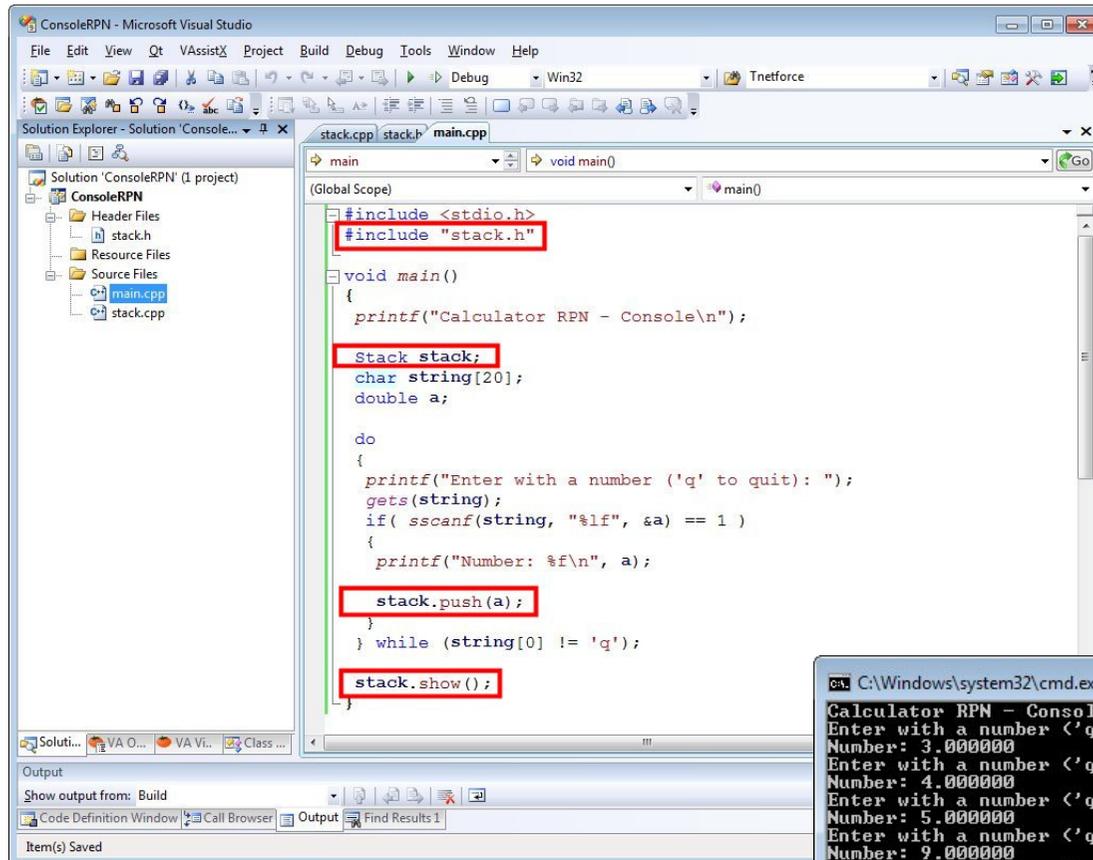
void Stack::push( double n )
{
    elems[top++] = n;
}

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

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

At the bottom right, a red arrow points from the 'Add' button in the dialog to the code editor, with the text: **Como utilizar a classe Stack no programa principal?**

Usando a classe Stack no programa principal para armazenar os dados



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

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

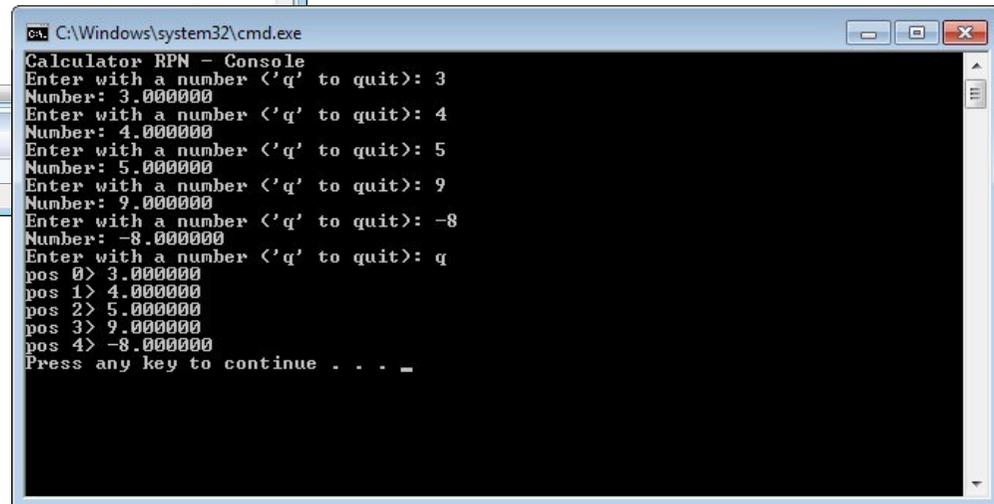
    Stack stack;
    char string[20];
    double a;

    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();
}
```

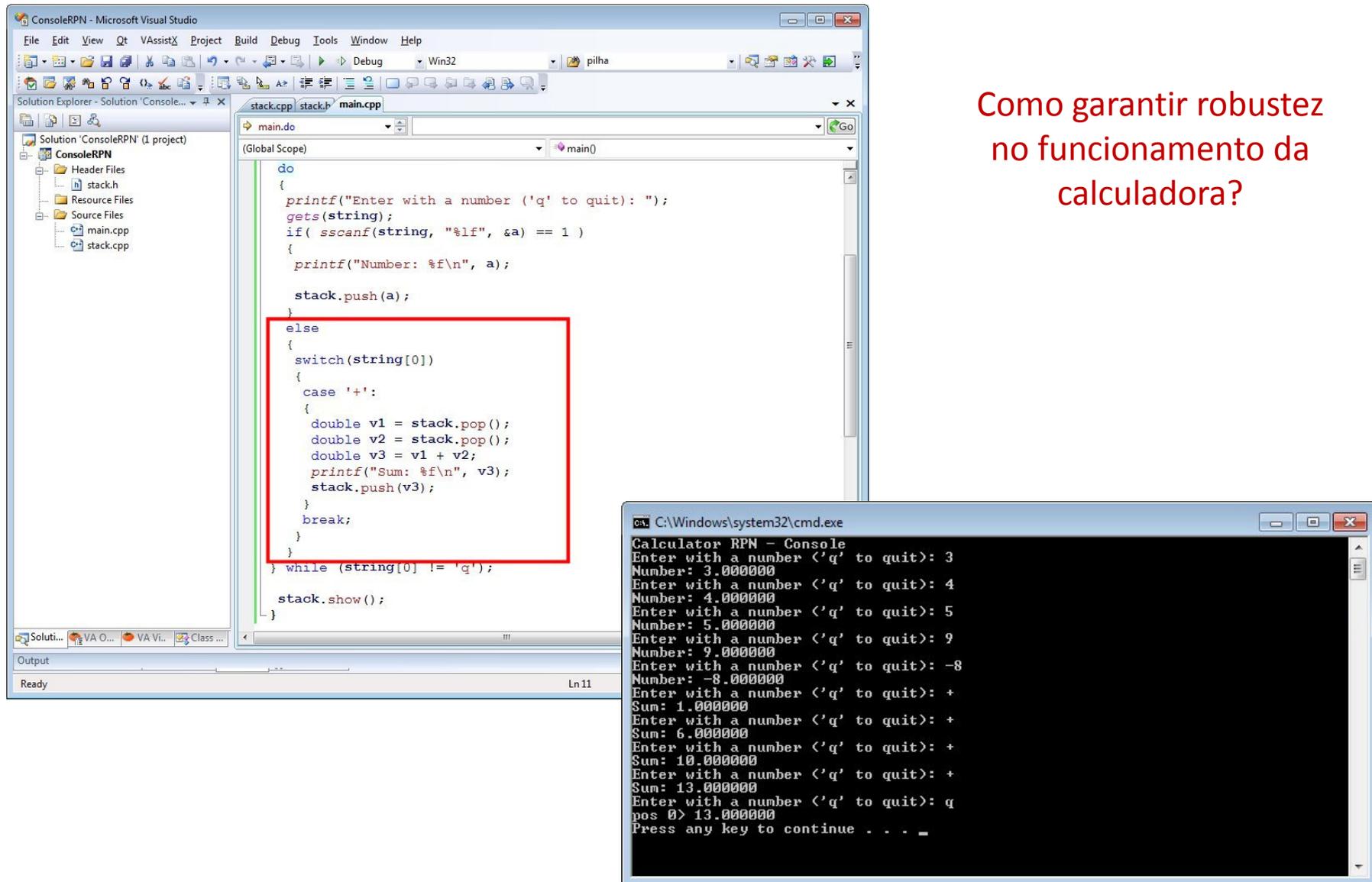
Como utilizar o objeto da classe Stack para realizar as operações da calculadora?



```
C:\Windows\system32\cmd.exe
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 . . . _
```

Implementação da operação de adição usando os dados da pilha

Como garantir robustez no funcionamento da calculadora?



The image displays a Visual Studio IDE window titled 'ConsoleRPN - Microsoft Visual Studio'. The Solution Explorer on the left shows a project named 'ConsoleRPN' with files 'stack.h', 'stack.cpp', 'main.cpp', and 'main.do'. The main editor shows the code in 'main.cpp' with a red box highlighting the addition logic:

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

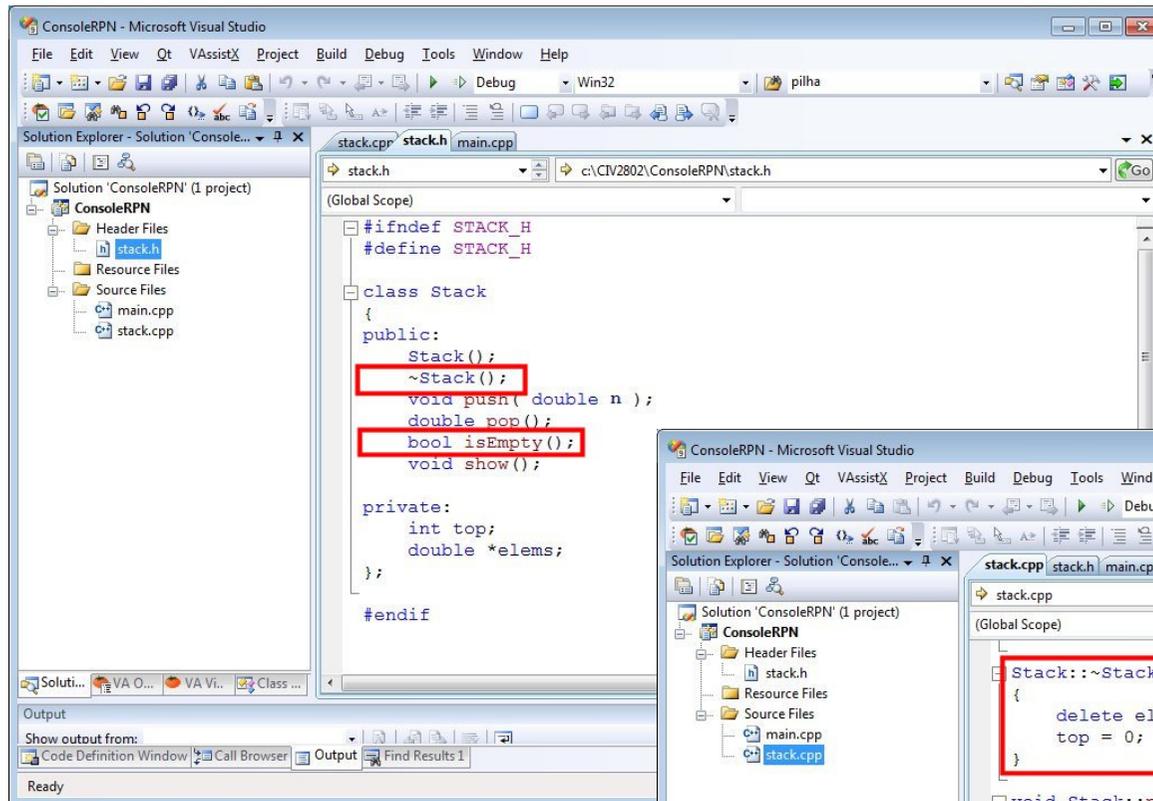
        stack.push(a);
    }
    else
    {
        switch(string[0])
        {
            case '+':
            {
                double v1 = stack.pop();
                double v2 = stack.pop();
                double v3 = v1 + v2;
                printf("Sum: %f\n", v3);
                stack.push(v3);
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

stack.show();
}
```

The console output window shows the execution of the program:

```
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 . . . _
```

Verificação de pilha vazia e eliminação de memória utilizada



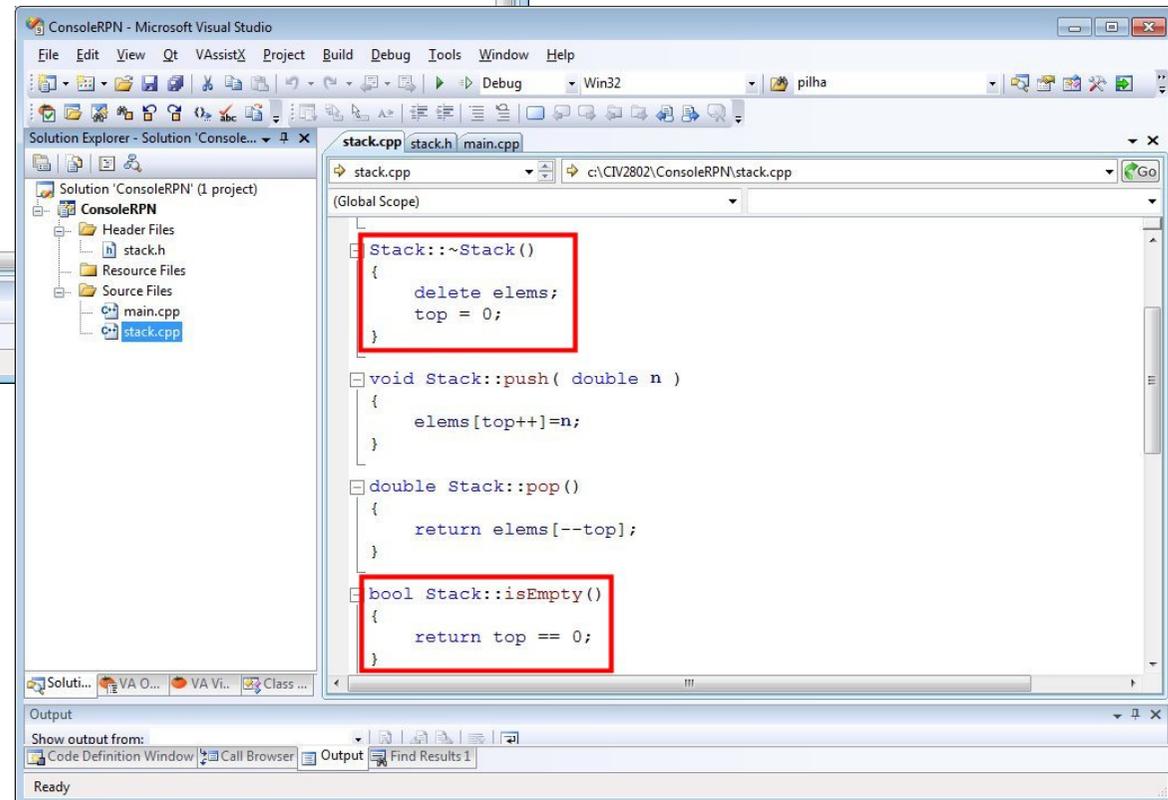
```
#ifndef STACK_H
#define STACK_H

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

private:
    int top;
    double *elems;
};

#endif
```

Quais tipos de problema podem ocorrer na execução e como evitá-los?



```
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;
}
```

Função auxiliar para obter os dois operandos de uma operação com o tratamento de possíveis erros

```
#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");
}
```

Quais são os mecanismos de passagem de parâmetros para função na linguagem C/C++?

Mecanismos de passagem de parâmetros para função em C/C++

Considere o programa abaixo com uma função “swap” cujo objetivo é a troca de valores de dois números inteiros.

```
#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 );
}
```

Escreva o resultado do programa, isto é, o que é impresso pelo programa?
Justifique.

Mecanismos de passagem de parâmetros para função em C/C++

Considere o programa abaixo com uma função “swap” cujo objetivo é a troca de valores de dois números inteiros.

```
#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 );
}
```

Resultado do programa:

a: 2 b: 5

Observa-se que não houve troca dos valores dos dois números.

O motivo é que o único mecanismo de passagem de parâmetro para função em C/C++ é por valor.

Neste mecanismo, é passada uma cópia do valor da variável para o parâmetro.

A função “swap” está trocando apenas os valores das cópias e não os valores das variáveis “a” e “b”.

A solução é simular uma passagem de parâmetro por referência. Isso é feito, passando (por valor) o endereço das variáveis. Os parâmetros da função “swap” passam a ser ponteiros para inteiros.

Programa corrigido:

```
#include <stdio.h>

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

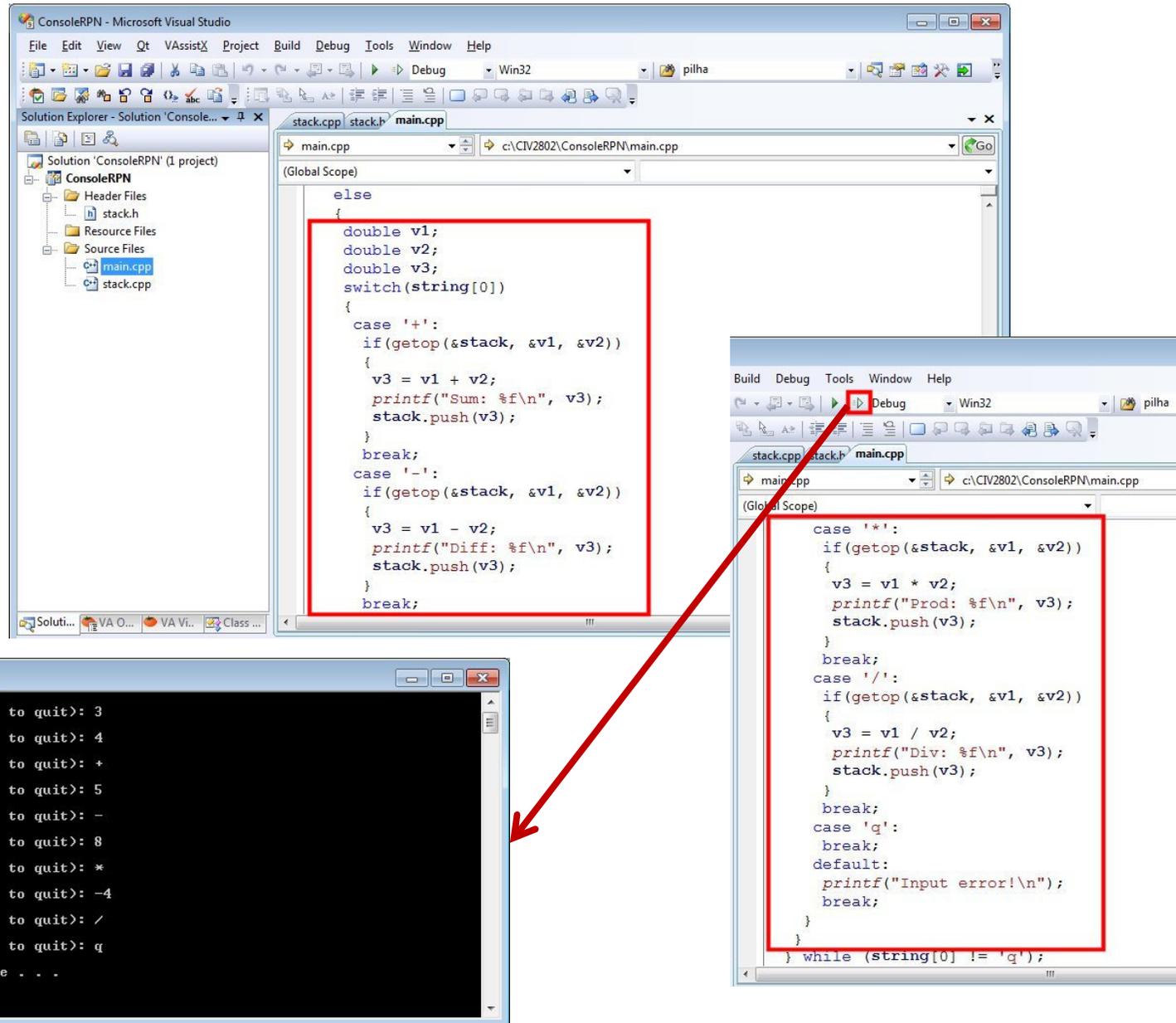
    temp = *px;
    *px = *py;
    *py = temp;
}

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

    swap( &a, &b );

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

Implementação das outras operações na calculadora via console



Testando o tratamento de possíveis erros na execução do programa

