

Programa “ConsoleRPN”

Programa criado com o Visual Studio Community 2013
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 Community 2013.

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

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 Zwięksakul*.) 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 * /

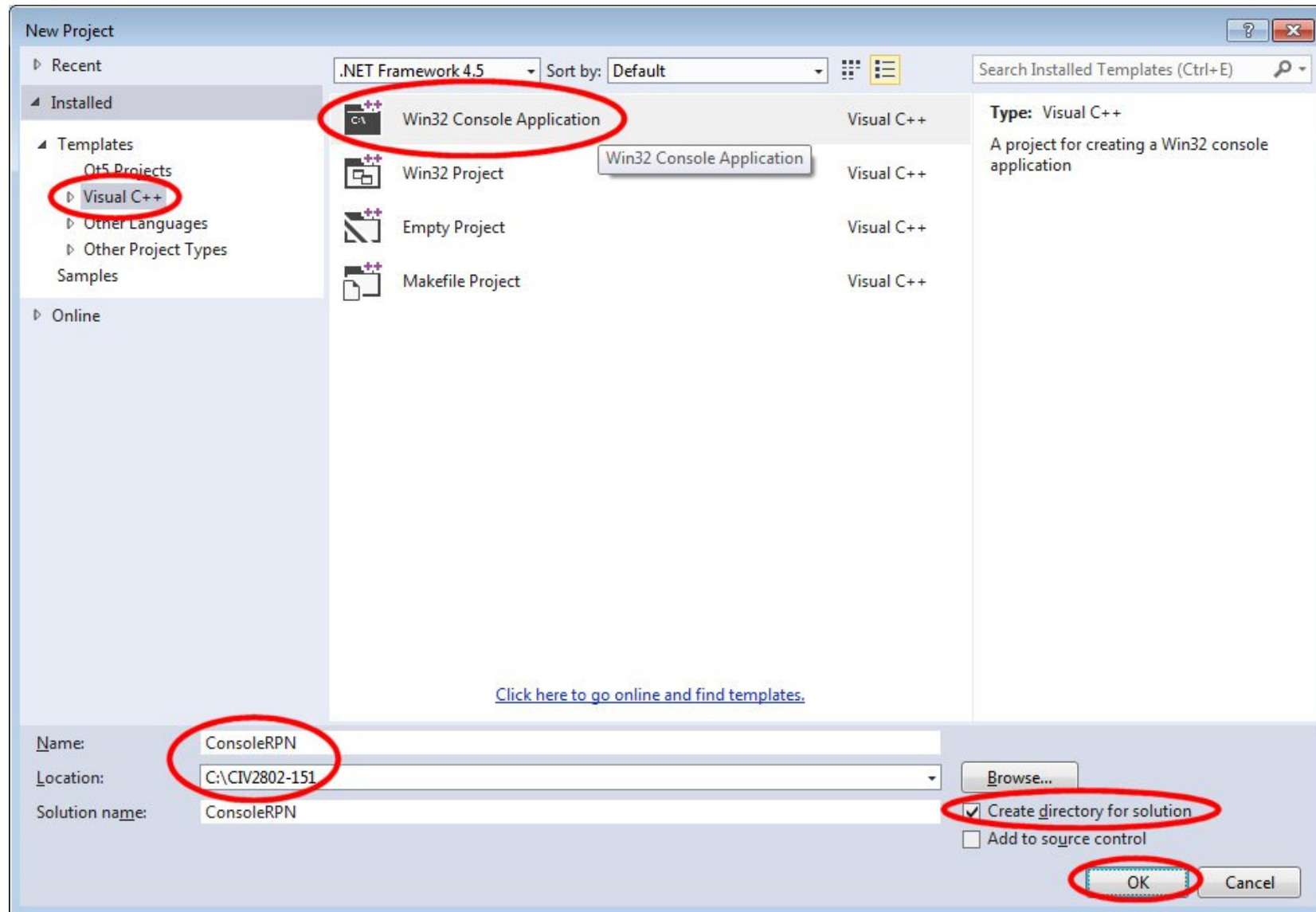
Exemplo clássico de calculadora que utiliza a Notação Polonesa Reversa (RPN)



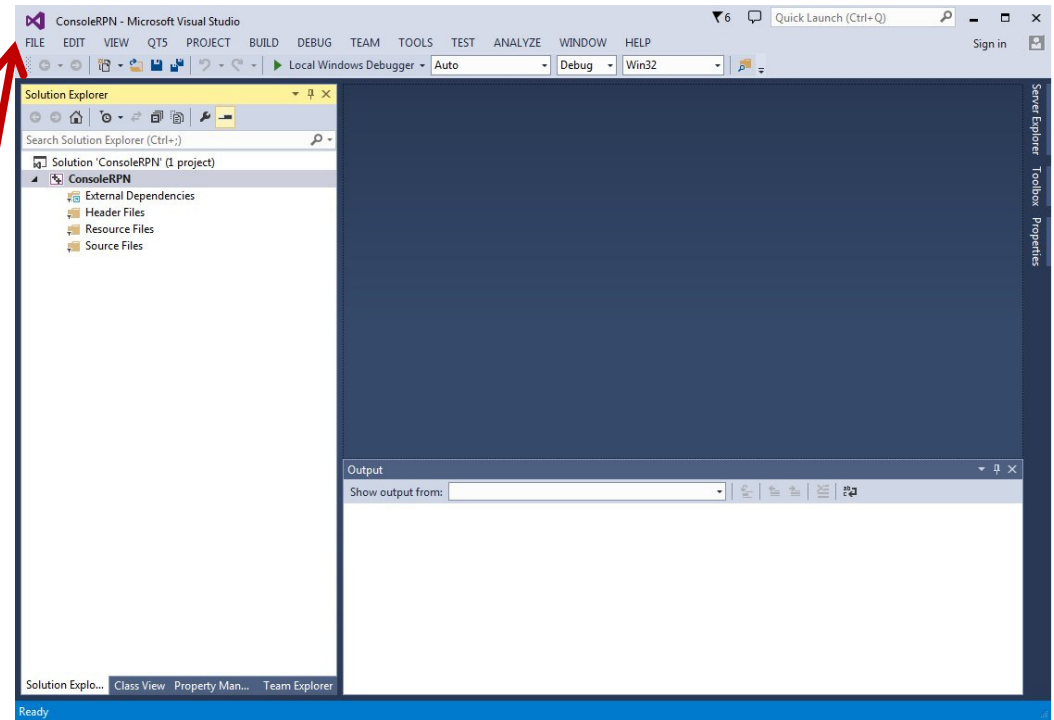
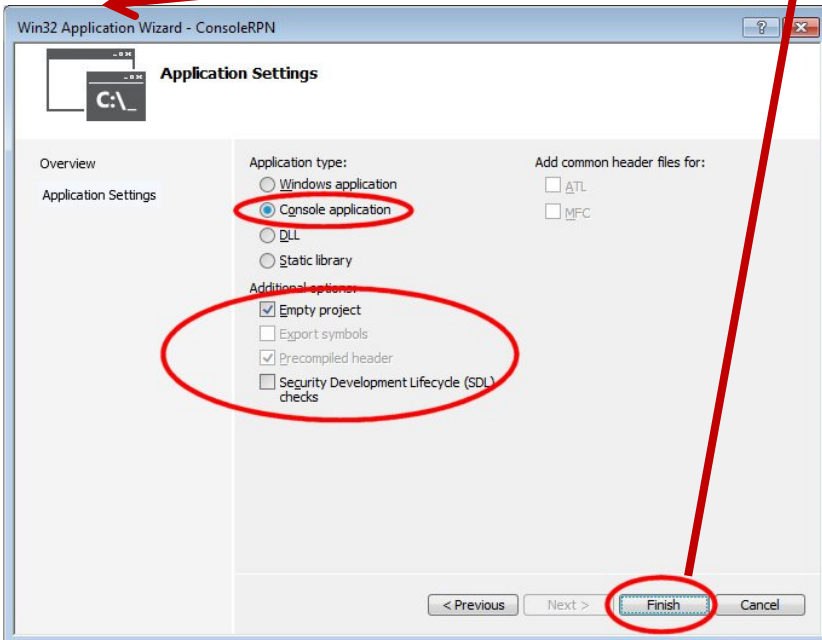
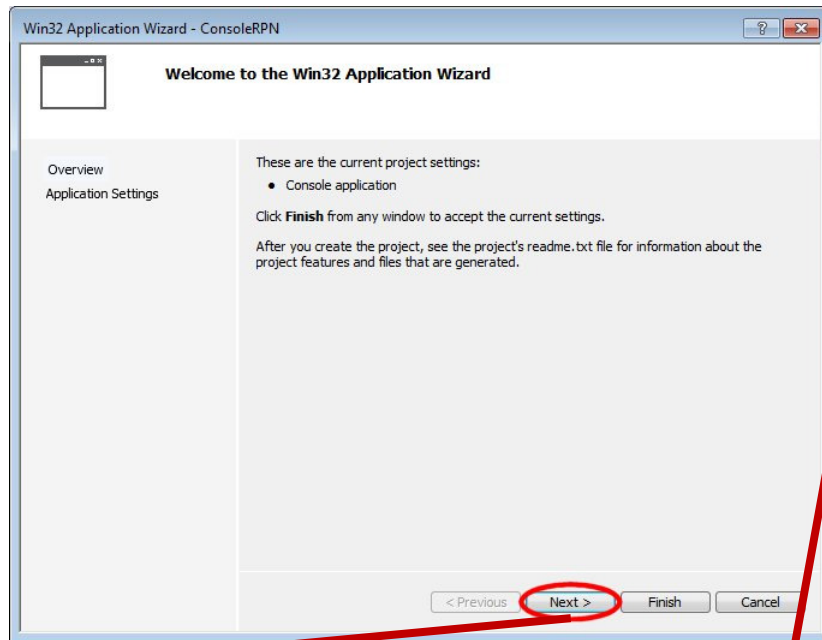
Como criar nossa própria calculadora utilizando uma linguagem de programação e uma interface gráfica?

Criação de um novo projeto no Visual Studio 2013 do tipo Console

Vamos começar programando uma calculadora sem interface gráfica.

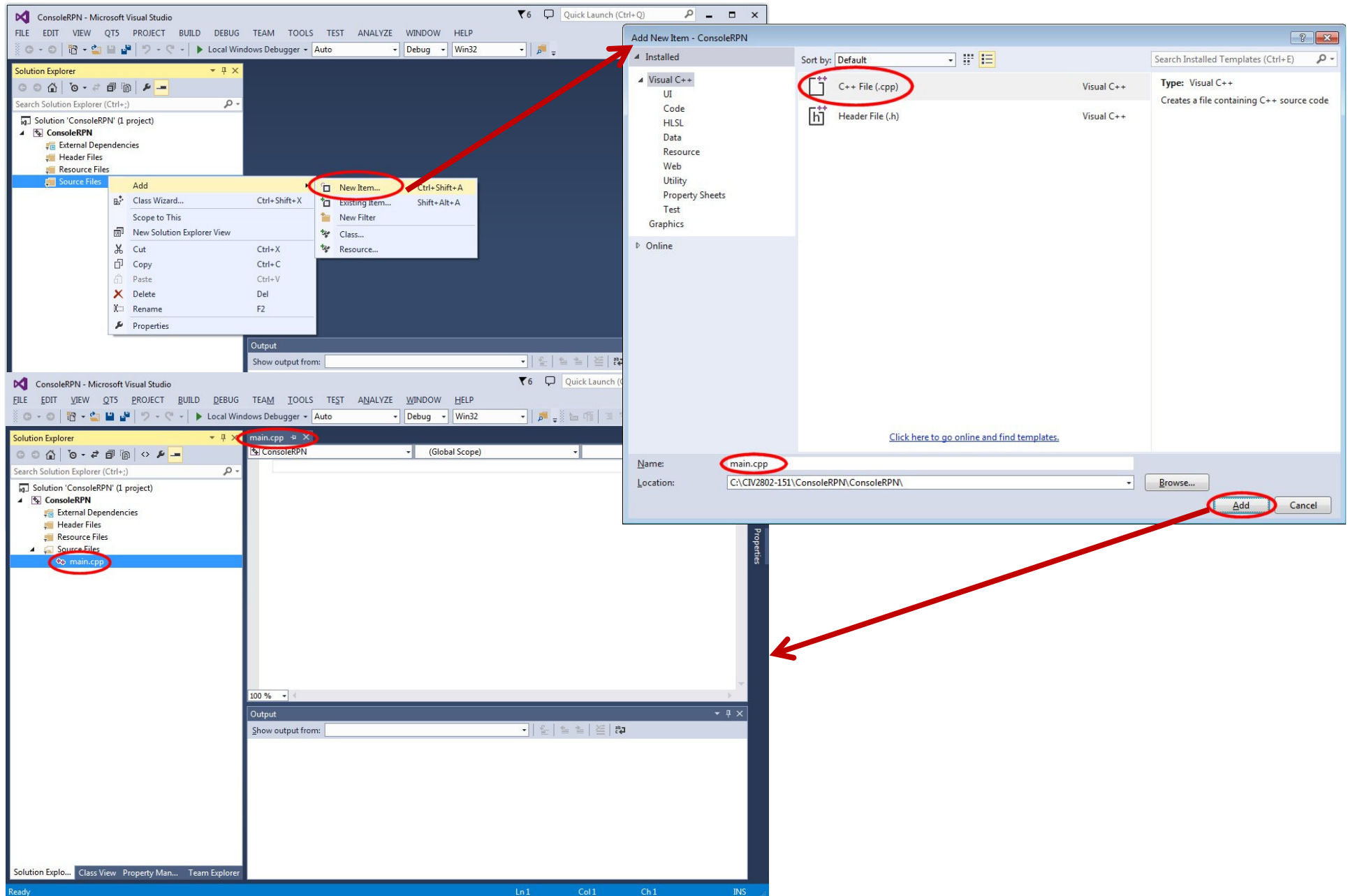


Wizard para criação de um projeto do tipo Console “Vazio”



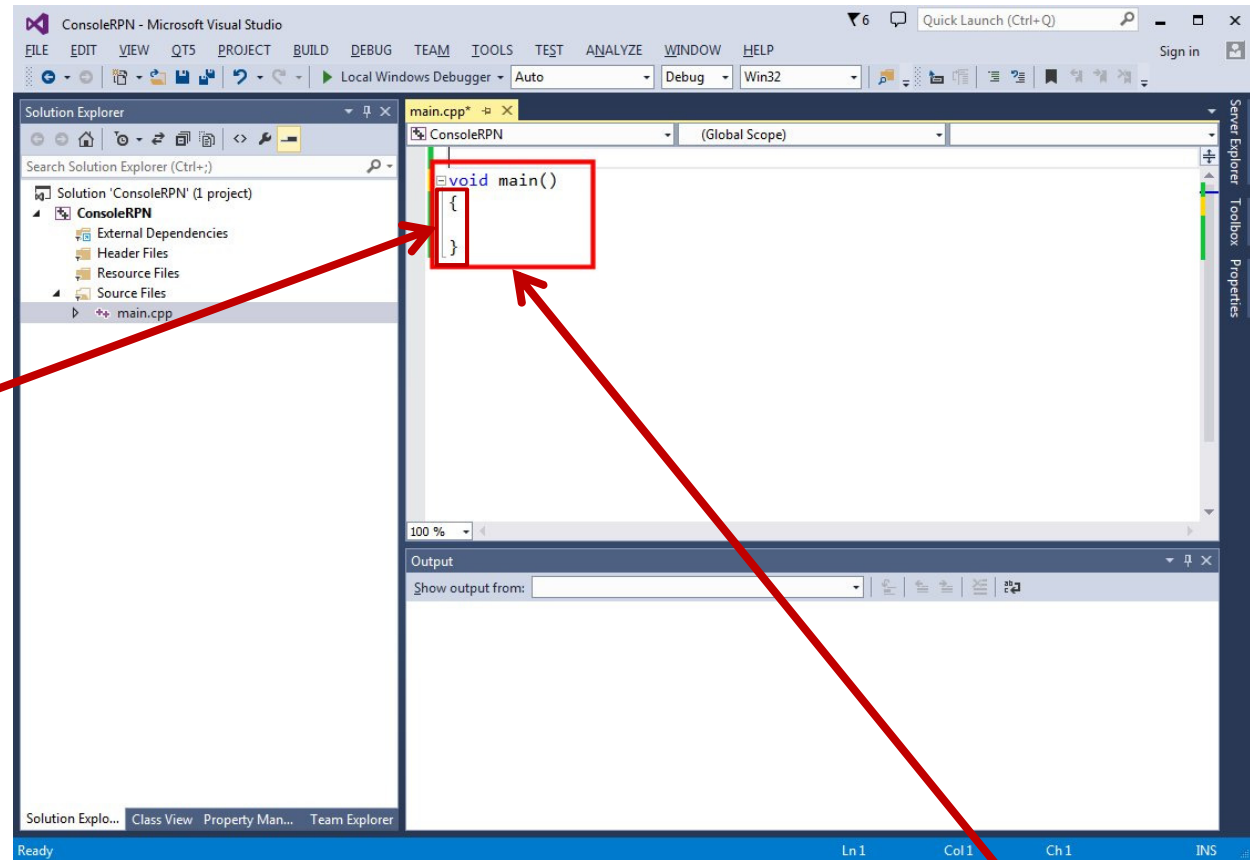
Como começar desenvolver um programa do zero?

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)`

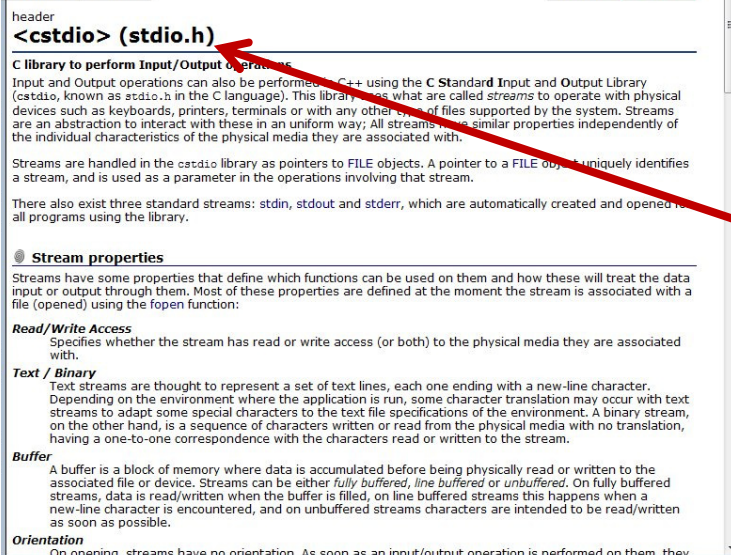
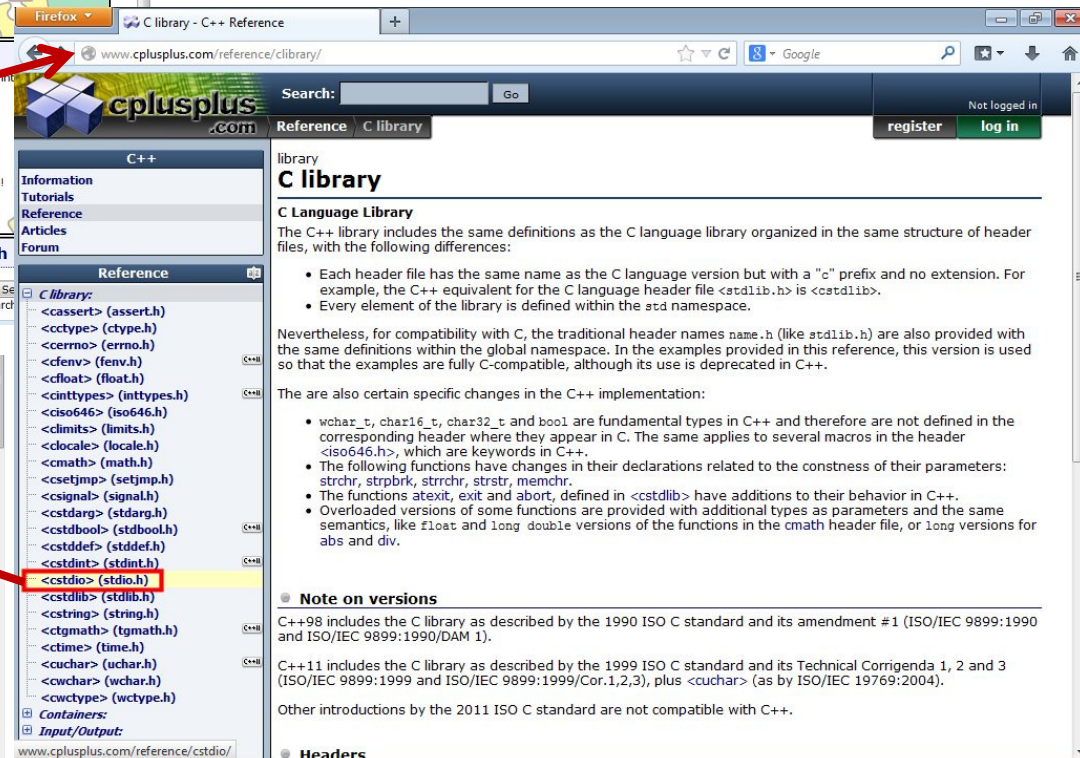
Todo programa em C contém pelo menos uma função:

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?



Tratamento de entrada e saída (io) em um programa em C

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, 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 a uniform way; All streams have similar properties independently the individual characteristics of the physical media they are associated with.
Streams are handled in the `stdio` 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.

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)
`sprintf` 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)
`vscanf` Read formatted data from stream into variable argument list (function)
`vprintf` Print formatted data from variable argument list to stdout (function)
`vscanf` Read formatted data into variable argument list (function)
`vsprintf` Write formatted data from variable argument list to sized buffer (function)
`vsprintf` Write formatted data from variable argument list to string (function)
`vsscanf` 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
`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

```
1 /* printf example */  
2 #include <stdio.h>  
3  
4 int main()  
5 {  
6     printf ("Characters: %c %c \n", 'a', 65);  
7     printf ("Decimals: %d %d\n", 1977, 650000L);  
8     printf ("Preceding with blanks: %10d \n", 1977);  
9     printf ("Preceding with zeros: %010d \n", 1977);  
10    printf ("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);  
11    printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);  
12    printf ("Width trick: %d \n", 5, 10);  
13    printf ("%s \n", "A string");  
14    return 0;  
15 }
```

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

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

The image shows a screenshot of Microsoft Visual Studio with two overlapping windows. The top window displays the source code for `main.cpp` in the `ConsoleRPN` project. The code includes the standard C I/O library and prints a message to the console. The bottom window shows the `BUILD` menu with `Build Solution` selected, and the `Output` window showing the successful build of the program.

```
#include <stdio.h>

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

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

\n produz uma nova linha no console.

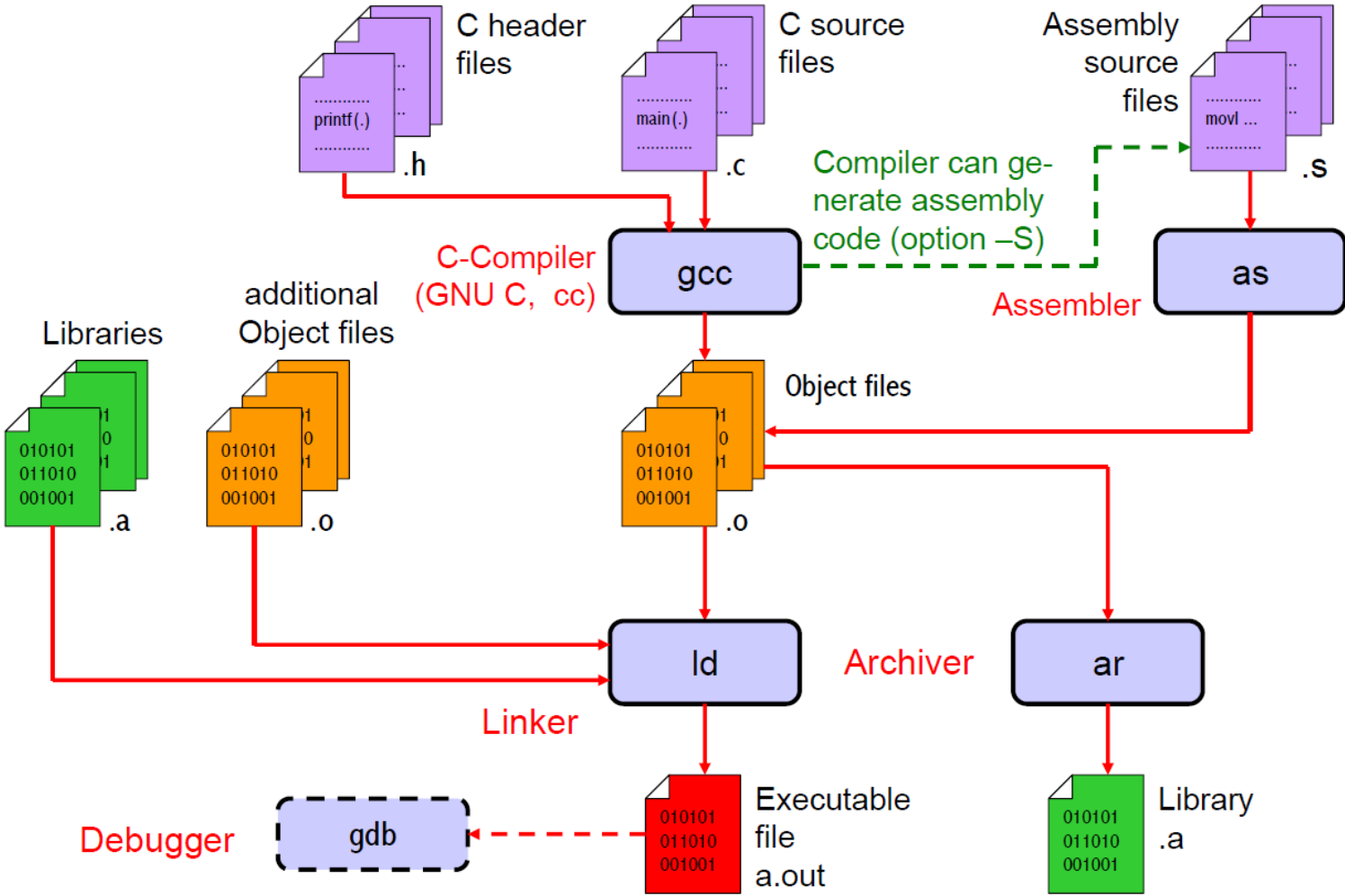
O que acontece quando o programa é construído (compilado e linkado)?

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 (`}`).

Build succeeded

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

The image shows a composite of three screenshots from Microsoft Visual Studio. The top-left screenshot displays the source code for a C++ program in `main.cpp`. The code includes `<stdio.h>` and defines a `main()` function. Inside `main()`, it prints "RPN Calculator - Console", declares three local variables: `double a = 10.5;`, `double b = 13.3;`, and `double c = a + b;`, and finally prints the sum: `printf("Sum: %f\n", c);`. A red box highlights these three lines, with an arrow pointing to a text box that says "Variáveis locais, apenas disponíveis no escopo de main()". The top-right screenshot shows the `BUILD` menu with `Build Solution` (F7) and `Rebuild Solution` (Ctrl+Alt+F7) highlighted. The bottom-right screenshot shows the `DEBUG` menu with `Start Without Debugging` (Ctrl+F5) highlighted. At the bottom, a console window shows the program's output: "RPN Calculator - Console", "Sum: 23.800000", and "Press any key to continue . . .".

```
#include <stdio.h>

void main()
{
    printf("RPN Calculator - Console\n");
    double a = 10.5;
    double b = 13.3;
    double c = a + b;
    printf("Sum: %f\n", c);
}
```

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

Build Solution (F7)
Rebuild Solution (Ctrl+Alt+F7)

Start Without Debugging (Ctrl+F5)

C:\Windows\system32\cmd.exe
RPN Calculator - Console
Sum: 23.800000
Press any key to continue . . .

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 image shows a screenshot of Microsoft Visual Studio with a C++ project named 'ConsoleRPN'. The main.cpp file is open, showing the following code:

```
#include <stdio.h>

double sum(double n1, double n2);

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

Annotations with red boxes and arrows point to specific parts of the code:

- Declaração (prototype) da função sum**: Points to the line `double sum(double n1, double n2);`.
- Chamada da função sum**: Points to the line `double c = sum(a, b);`.
- Um par de `/* */` define um comentário que é ignorado pelo compilador**: Points to the multi-line comment block above the function definition.
- Implementação da função sum**: Points to the function definition `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 signature.
- A função sum retorna um valor double**: Points to the `double` return type in the function signature.

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("RPN Calculator - 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);
}
```

The code is partially enclosed in a red box. Below the code editor, the 'Output' window shows the program's execution:

```
C:\Windows\system32\cmd.exe
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("RPN Calculator - 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
RPN Calculator - 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

The screenshot shows the Portuguese Wikipedia page for 'Estrutura de dados'. The page title is 'Estrutura de dados' and it includes a search bar, navigation tabs (Artigo, Discussão), and a sidebar with various links like 'Página principal', 'Eventos atuais', and 'Colaboração'. The main content area starts with the definition: '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.' It then discusses different types of data structures and their applications, mentioning B-trees and hash tables. A diagram of a binary tree is shown, with nodes containing numbers: root 2, left child 7, right child 5, 7's left child 2, 7's right child 6, 5's left child 9, 6's left child 5, and 6's right child 11. Below the diagram is a caption: 'Uma árvore binária é uma estrutura de dados.'

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

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.

A Estrutura de Dados Pilha (*Stack*)

Firefox - W Pilha (informática) - Wikipédia, a encicl... +

pt.wikipedia.org/wiki/Pilha_(informática) pilha

WIKIPÉDIA
A enciclopédia livre

Artigo Discussão Ler Editar Editar código-fonte Ver histórico Pesquisa

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 [editar | editar código-fonte]

Representação simples de uma pilha.

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

Programação Orientada a Objetos

The screenshot shows the Portuguese Wikipedia page for "Orientação a objetos". The browser is Firefox, and the URL is pt.wikipedia.org/wiki/Orientação_a_objetos. The page features a navigation menu on the left, a main content area, and a sidebar on the right.

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
Correlatos

Noutras línguas
Afrikaans
العربية
Azərbaycanca
Беларуская
Беларуская (тарашкевіца)

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, acadêmico — [Sciur](#) — [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 header file for a stack in Visual Studio. It is divided into three main sections:

- Top Panel:** Shows the Visual Studio interface with the Solution Explorer on the left. The 'Add' menu is open, and 'New Item...' is selected. A red circle highlights the 'New Item...' option.
- Middle Panel:** Shows the 'Add New Item - ConsoleRPN' dialog box. The 'Visual C++' category is expanded, and 'Header File (.h)' is selected. The name 'stack.h' is entered in the 'Name' field. A red circle highlights the 'Header File (.h)' option and the 'Add' button at the bottom right.
- Bottom Panel:** Shows the code editor with the content of 'stack.h'. The code is enclosed in a red box:

```
#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 shows two screenshots of Microsoft Visual Studio. The top screenshot shows the 'Add New Item' dialog box with 'C++ File (.cpp)' selected. The bottom screenshot shows the 'stack.cpp' file being created and its code being implemented.

Top Screenshot: Add New Item Dialog

- Visual C++ > C++ File (.cpp) is selected.
- Name: stack.cpp
- Location: C:\CIV2802-151\ConsoleRPN\ConsoleRPN\
- Buttons: Add, Cancel

Bottom Screenshot: Code Implementation

```
#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]);
    }
}
```

Como utilizar a classe Stack no programa principal?

Usando a classe Stack no programa principal para armazenar os dados

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

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

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

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

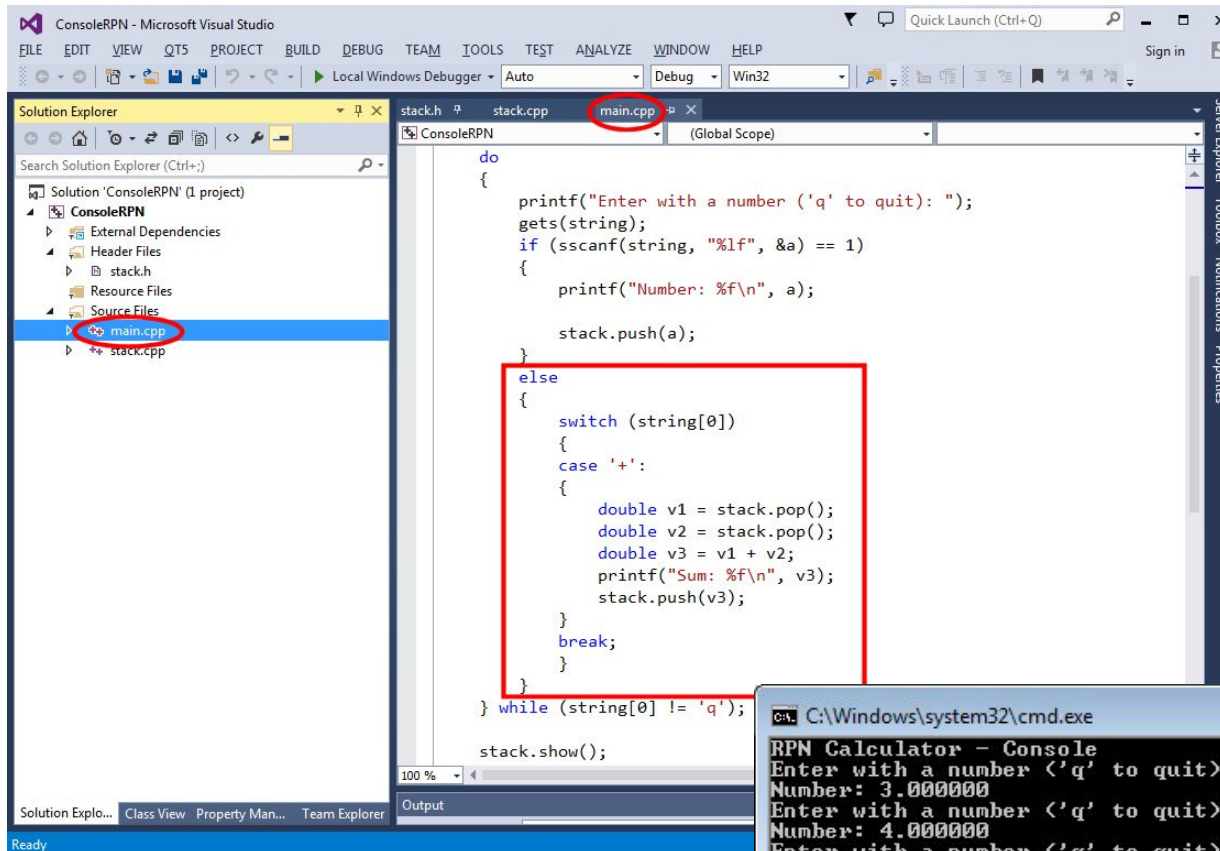
    do
    {
        printf("Enter with a number ('q' to quit): ");
        gets(string);
        if (sscanf(string, "%lf", &a) == 1)
        {
            printf("Number: %f\n", a);

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

    stack.show();
}
```

```
C:\Windows\system32\cmd.exe
RPN Calculator - 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 . . . _
```

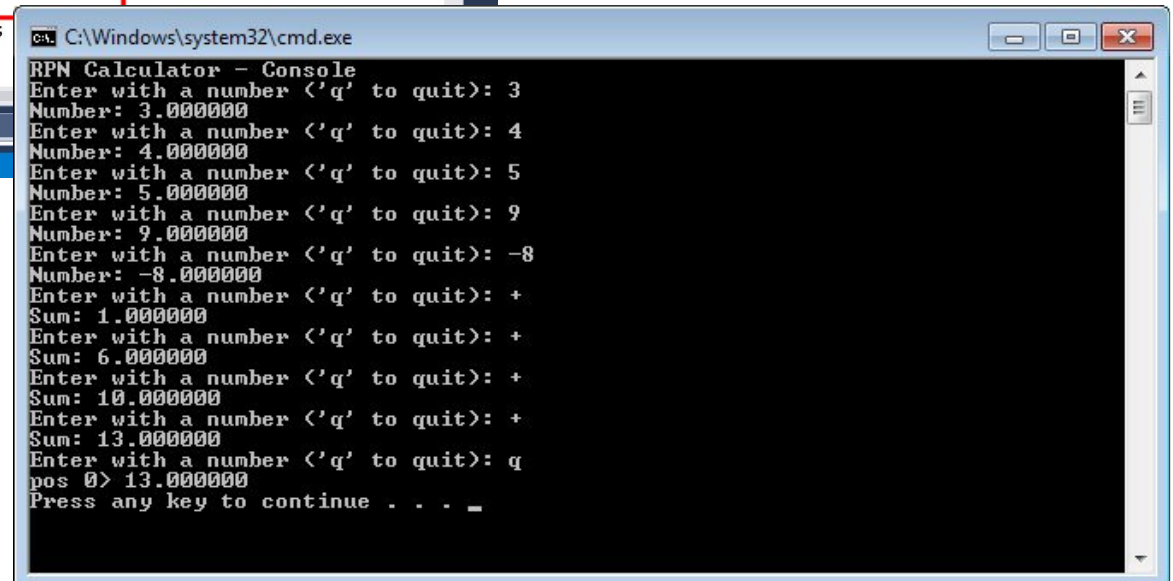
Implementação da operação de adição usando os dados da pilha



```
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();
```

Como garantir robustez no funcionamento da calculadora?



```
RPN Calculator - 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

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

This screenshot shows the Visual Studio IDE with the 'stack.h' header file open. The Solution Explorer on the left shows the project structure with 'stack.h' selected. The code in the main editor defines a 'Stack' class with several methods. Red boxes highlight the constructor, destructor, and isEmpty method declarations.

```
#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
```

This screenshot shows the Visual Studio IDE with the 'stack.cpp' implementation file open. The Solution Explorer on the left shows 'stack.cpp' selected. The code implements the Stack class methods. Red boxes highlight the destructor, isEmpty method, and show method implementations.

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

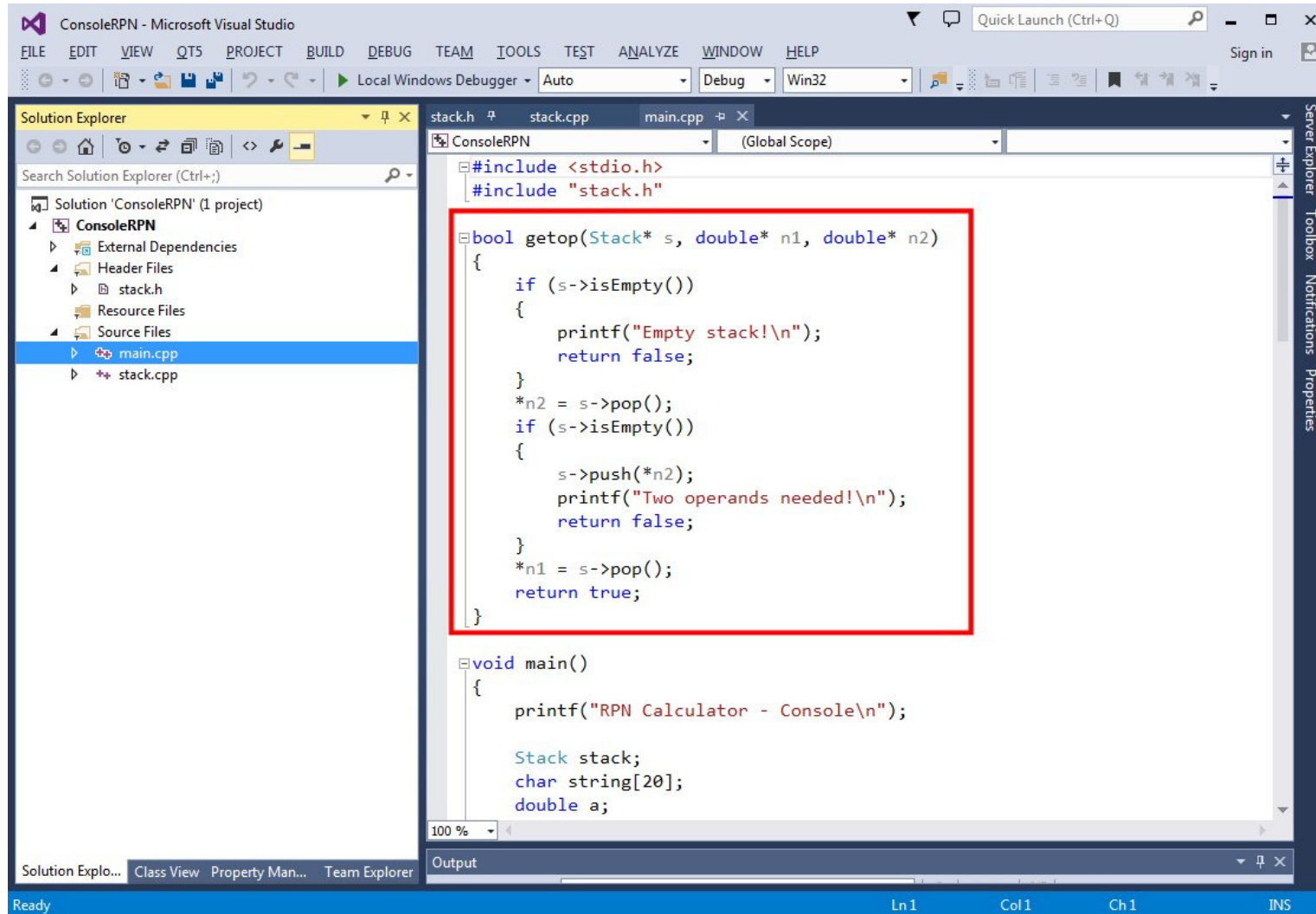
void Stack::push(double n)
{
    elems[top++] = n;
}

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

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

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

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("RPN Calculator - Console\n");

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

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

The image displays the implementation of a Reverse Polish Notation (RPN) calculator in C++ using Visual Studio. The code is organized into two screenshots of the IDE, with a terminal window at the bottom showing the program's execution.

Visual Studio Screenshot 1 (Left): Shows the code for addition and subtraction operations. The `main.cpp` file is selected in the Solution Explorer. The code includes a `switch` statement with two cases: `case '+':` and `case '-':`. Both cases use `getop(&stack, &v1, &v2)` to retrieve two numbers from the stack, perform the operation, and push the result back onto the stack.

```
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;
    }
}
```

Visual Studio Screenshot 2 (Right): Shows the code for multiplication and division operations. The `main.cpp` file is selected in the Solution Explorer. The code includes a `switch` statement with two cases: `case '*':` and `case '/':`. Both cases use `getop(&stack, &v1, &v2)` to retrieve two numbers from the stack, perform the operation, and push the result back onto the stack. A `default` case handles input errors.

```
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');
```

Terminal Window: Shows the execution of the RPN Calculator. The user enters numbers and operators, and the program outputs the results of the operations.

```
RPN Calculator - 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 . . . _
```


Testando o tratamento de possíveis erros na execução do programa

The image displays two screenshots of Microsoft Visual Studio and a Windows command prompt window. The top-left screenshot shows the Visual Studio IDE with the 'main.cpp' file open. A red box highlights a section of code within a switch statement, specifically the '+' and '-' cases. The code includes variable declarations for v1, v2, and v3, and logic for pushing values onto a stack and printing the sum or difference. The top-right screenshot shows the same IDE with the '*' and '/' cases highlighted in a red box, including logic for multiplication and division. The bottom screenshot is a command prompt window titled 'RPN Calculator - Console' showing the program's execution. It prompts the user to enter numbers and operators. The user enters '3', '4', '+', and 'q'. The program outputs 'Number: 3.000000', 'Number: 4.000000', 'Sum: 7.000000', and 'Two operands needed!' before returning to the prompt. The prompt then shows 'pos 0 > 7.000000' and 'Press any key to continue . . .', indicating the program has finished execution.

```
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');
```

```
C:\Windows\system32\cmd.exe
RPN Calculator - 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 . . .
```