Sumário

1. Objetivo

Realizar a introdução ao uso do Visual Studio Community Edition C/C++ (VSCE2017) e do Bloodshed Dev-C++. Apresentar comandos básicos da linguagem C e apresentar a estrutura de seleção (if..). Implementar alguns programas básicos escritos na linguagem C. Apresentar um paralelo entre a linguagem C/C++ e o português estruturado (PE),

2. Instalação do Software

2.1. Instalação VSCE2017

Para baixar e instalar o VSCE2017 no computador (é gratuito), visite o site abaixo.

https://docs.microsoft.com/pt-br/visualstudio/install/install-visual-studio

Este IDE (Ambiente de Desenvolvimento Integrado) com compilador C/C++ será utilizado no laboratório. A seguir, alternativas para quem desejar. Ele já está instalado no laboratório da Uninter.

2.2. Instalação Dev-C++

O Bloodshed DEV-C++ é um bom IDE para Windows com um compilador C/C ++, que é instalado de uma só vez. Pesquisar na Web ou tentar http://www.bloodshed.net/download.html para instalar em seu computador pessoal. Ele já está instalado no laboratório da Uninter.

2.3. Alternativas de Compiladores C/C++

2.3.1. jGRASP

IDE desenvolvido pelo Departamento de Ciência da Computação e Engenharia de Software na Faculdade de Engenharia Samuel Ginn da Universidade de Auburn (www.auburn.edu). É um ambiente de desenvolvimento leve criado especificamente para fornecer geração automática de visualizações de software para melhorar a compreensão do software,

uma questão fundamental para iniciantes. É gratuito. O jGRASP é executado no Windows, Mac OS X e Linux (UNIX e outros) com o Java Virtual Machine (Java versão 1.5 ou superior). O jGRASP gera CSDs (Control Structure Diagrams, ou Diagramas de Estrutura de Controle, que lembram fluxogramas) para as linguagens de programação com as quais trabalha, que são Java, C, C++, Objective-C, Python, Ada e VHDL.

2.3.2. XCode para MAC OS X

É um IDE junto com um Compilador C/C ++, que é instalado de uma só vez (pesquisar na Web ou tentar https://developer.apple.com/xcode/).

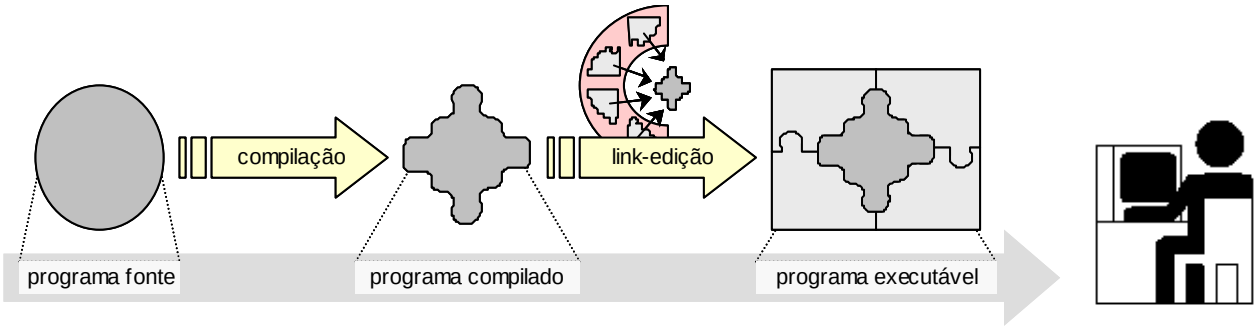
2.3.3. Para Linux

A maioria dos IDEs requer uma instalação prévia de um compilador C/C ++. Bons IDEs: Code::Blocks (www.codeblocks.org), KDevelop (www.kdevelop.org) ou CodeLite (www.codelite.org), entre outros.

3. BASE TEÓRICA

3.1. Tipos de Programas

Um programa em C é formado por uma ou mais funções escritas na sintaxe dessa linguagem que, em conjunto, formam o *programa fonte* (ou *código fonte*).



Para transformar o programa fonte em um *programa executável* pelo computador, é necessário traduzi-lo para a linguagem de máquina através do processo de compilação[1]. Nesse processo, o compilador primeiramente verifica a sintaxe do programa fonte antes de realizar a tradução. Feito isso, tem-se como resultado dessa tradução o *programa compilado* (extensão *.obj*).

Para transformar o programa compilado no programa executável (extensão *.exe*) é necessário realizar o processo de link-edição, que irá ligar o programa compilado a bibliotecas do C (como stdio.h) que estão sendo utilizadas e resolver outras referências. Após esse último passo, o programa está pronto para ser executado ou "rodado".

3.2. Um Programa em C

O menor programa possível de se escrever em C é o mostrado ao lado (o programa "vazio": não realiza nada).

```
int main(){}
```

A palavra *main* é o nome da função principal do programa em C, o qual pode ser constituído de uma ou mais funções, tendo, no mínimo, a função *main()*, pois quando o programa é executado, ele inicia por essa função.

Toda função é seguida por um par de parênteses, com ou sem uma lista de parâmetros (dados para a realização de algum algoritmo). A função *main()* acima não recebe nenhum parâmetro, o que será o padrão de todos os programas a serem desenvolvidos (a função *main()* pode receber parâmetros se for declarada para isto, mas isso envolve outros tipos de aplicação que não serão vistos aqui).

Sendo *main()* uma função, ela deve ter um tipo de retorno (inteiro, real, caractere, etc.). O padrão utilizado é definir o valor de retorno como sendo *int* (inteiro). Como o valor de retorno da função *main()* muitas vezes não é utilizado, alguns ambientes de programação em C permitem declará-la como sendo do tipo *void* ("vazio", "inexistente"), que significa que a função *main()* não tem um valor de retorno. Mas isso não é recomendado por questões de portabilidade e compatibilidade entre plataformas.

O corpo de uma função em C (declarações e comandos) está contido entre um par de chaves ("{" e "}"). No exemplo acima, a função *main()* não realiza nada, pois não há nenhum comando entre as chaves.

Genericamente falando, a forma mais comumente encontrada para um programa C com apenas a função *main()* pode ser vista na figura ao lado. Alguns compiladores exigem que o valor de retorno seja do tipo *int* (ver item 3.3) devido a aderência a padrões (como o C11), mas o VSCE2017 permite o uso do tipo de retorno *void* para a função *main()*.

```
int main() {
  <declarações;>
  <comandos;>
}
```

[1] Existe um processo de pré-compilação, que estende o código fonte original com as definições introduzidas principalmente pelas diretivas de compilação. Isso será comentado em momento oportuno.

Exemplo 1. Um programa em C.

```
#include "stdio.h"   /* Essa diretiva inclui o arquivo de cabeçalho onde está definida
                        a função printf(), usada no programa. No VSCE2017, "stdio.h" é
                        substituído por " stdafx.h". */
int main() {
   int N; // Declaração da variável N como sendo do tipo inteiro.
   N=20;  // Atribuição de valor.
   printf("Eu tenho %d alunos.\n",N); // Imprime um texto formatado, onde %d indica
                                      // que naquela posição impresso um valor inteiro,
                                      // que é o valor da variável N.
}
```

Resultado do programa

```
Eu tenho 20 alunos.
```

Nota. *Comentários na linguagem C padrão são colocados sempre entre os símbolos **/*** e *****/**; no C++ também é aceito o símbolo //, que deve ser colocado apenas no início do comentário, sendo desnecessário repeti-lo ao final deste, desde que o comentário esteja em uma só linha.*

No exemplo anterior há um elemento adicional: a diretiva de inclusão *#include*. Toda vez que for necessário utilizar uma função da biblioteca padrão do C, é preciso incluir a diretiva de inclusão correspondente no início do texto do programa (e antes da definição das funções) dizendo o nome do arquivo de cabeçalho (*nome.h*) onde a função a ser utilizada está definida. O mais comum será a inclusão de "stdlib.h" ("stdafx.h" no VSCE) para usar as funções printf() e scanf(), que correspondem, respectivamente, aos comados imprima() e leia() do PE, e "math.h", para usar as funções matemáticas (sin, cos, log, exp, etc.).

Exemplo 2. Função printf().

Se um programa utiliza as funções *printf()* (definida em stdio.h) e *log()* (definida em math.h), ele deve ter um código como definido abaixo.

```
#include "stdio.h"   // Para usar printf(). No VSCE usa-se stdafx.h.
#include "math.h"    // Para usar log(), logaritmo neperiano.

void main() {
     : /* Outros comandos. */
  double X;  /* X é do tipo real de dupla precisão. */
  X = 3.5;
  printf("O log neperiano de 3.5 eh %f.", log(X)); /* Imprime ln(3.5). */
     : /* Outros comandos. */
}
```

Resultado do trecho de programa

```
O log neperiano de 3.5 eh 1.252763.
```

3.3. Alguns Tipos do C

A linguagem C possui uma série de tipos. Na tabela abaixo estão alguns exemplos.

Tipo Básico	Tipo do C	Bytes	Outros Nomes	Faixa de Valores
Inteiro	`int`	*	signed, signed int	dependente do sistema
	`unsigned int`	*	unsigned	dependente do sistema
	`short`	2	short int, signed short int	-32.768 até 32.767
	`unsigned short`	2	unsigned short int	0 até 65.535
	`long`	4	long int, signed long int	-2.147.483.648 até 2.147.483.647
	`unsigned long`	4	unsigned long int	0 até 4.294.967.295
Real	`float`	4	-	3,4E +/- 38 (7 dígitos)
	`double`	8	-	1,7E +/- 308 (15 dígitos)
	`long double`	10	-	1,2E +/- 4932 (19 dígitos)

Um aspecto importante deve ser notado: cada tipo tem associado uma faixa de valores que consegue representar. Por exemplo, o tipo *short* não pode ser utilizado para definir uma variável inteira que possa assumir valores como 45.000 (>32.767) ou -92.233 (<-32.768).

Outra particularidade é que esta faixa de valores, para alguns tipos, depende do sistema onde o compilador trabalha. O tipo *int* é um exemplo: pode ter 2 bytes de tamanho em determinado computador, e 4 bytes em outro (byte é uma quantidade que, resumidamente, representa um código de caractere, ou um número, variando entre 0 e 255). Atualmente, o mais comum é 4 bytes devido a maior capacidade das máquinas. O tipo short utiliza dois bytes, podendo então representar 256 x 256 = 65536 valores: -32768 a 32767.

3.4. Nome de Variáveis em C

O nome de variáveis em C segue convenções semelhantes ao do PE:

- deve sempre começar com uma letra;
- pode conter dígitos (0,1,..,9), letras (a,..., z, A,.., Z) e alguns caracteres especiais (_, $, etc.); exemplo: x, y, v00, primeiro_nome, endereco, contador, din$, etc;
- não podem ser utilizados ç ou caracteres acentuados (á, ã, é, etc.);
- não pode conter espaços em branco: um nome de variável como valor de x não é possível, mas valor_de_x é válido.

Uma particularidade importante relativa aos identificadores em C[2]: *identificadores com letras maiúsculas ou minúsculas são considerados diferentes*. Ex.: os identificadores *Contador* e *contador* são considerados como referências a duas variáveis distintas.

3.5. Declaração de Variáveis em C e Atribuição

A declaração de variáveis em C é bem semelhante ao que foi visto no PE. O símbolo de atribuição é o "=" (no PE é "←"), e o de comparação de igualdade é "==" (no PE é "="). Exemplos:

Código	Significado
`int k=0;`	Declaração de uma variável do tipo int (inteiro) chamada k. Atribuído o valor zero na declaração.

`float velocidade, distancia;`
`velocidade = 15.5;`
`distancia = 321.9;`
	Declaração de duas variáveis do tipo float (real): velocidade e distancia. Valores atribuídos após as declarações.

3.6. Operadores

Alguns dos operadores em C/C++ são mostrados na tabela abaixo.

Operador	Prioridade	Símbolo	Descrição	Observação
Aritmético	1	*, /	multiplicação, divisão (real ou inteira)	Se os dois operandos forem inteiros (divisão inteira), o resultado será inteiro (truncado se necessário). Exemplos: 5/2 = 2, mas 5.0/2 = 5/2.0 = 5.0/2.0 = 2.5
		%	resto da divisão inteira	Ex.: 5 % 2 = 1; 13 % 8 = 5; 43 % 2 = 3
	2	+, -	soma, subtração	Ex.: 5 + 3 = 7, 4 - 1.0 = -3.0
Relacional	3	>, >=	maior, maior ou igual	O resultado de uma expressão relacional é um valor do tipo lógico: (10 > 3) é .V. ; (3 == 5) é .F.; (3 != 5) é .V.
		<, <=	menor, menor ou igual	
	4	==, !=	igual, diferente (não igual)	
Lógico	5	!	não	-
	6	&&	e	-
	7	||	ou	-

***Nota**. O C não tem o tipo lógico. Ao invés disso, trabalha com zero (0) como sendo falso e um (1) como sendo verdadeiro, ou seja, valores inteiros. Dessa forma, é possível estabelecer a precedência entre operadores relacionais conforme mostrado acima.*

3.7. Funções de Entrada e Saída da Biblioteca Padrão do C

3.7.1. Função printf()

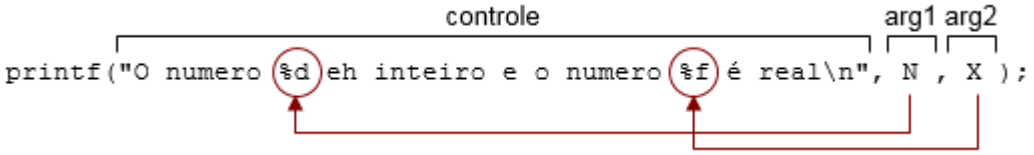
Essa função corresponde ao imprima() do PE. Está definida na biblioteca stdio.h (stdafx.h no VSCE). A forma geral da função *printf()* é a seguinte:

```
printf(controle, arg1, arg2, arg3, ...);
```

onde controle é a cadeia de caracteres a serem impressos com o respectivo controle da formatação do texto, e arg1, arg2, arg3, etc, são argumentos que serão impressos conforme definido em controle.

[2] Dê uma olhada no material sobre português estruturado para recordar a definição de identificadores. Eles são usados como o próprio nome diz: para identificar. Assim, nome de variáveis e de funções são identificadores.

A figura abaixo ilustra como ocorre a substituição dos argumentos na cadeia de controle (N é inteiro, e X é real).



Nessa figura tem-se:

- %d: indica que o valor de uma variável inteira (int) será colocado nesta posição (no caso, N);
- %f: indica que o valor de uma variável real (float) será colocada nesta posição (no caso, X);
- \n: faz com que o cursor passe para a próxima linha; podem ser usados vários.

Os exemplos da tabela abaixo auxiliam no entendimento das variações no uso de *printf()*.

Comando (para K = 3 e X = 2.346)	**Resultado** (_ significa espaço)
`printf("O numero [%d] eh inteiro", K);`	`O numero [3] eh inteiro`
`printf("O numero [%3d] eh inteiro", K);`	`O numero [_ _ 3] eh inteiro`
`printf("O numero [%-3d] eh inteiro", K);`	`O numero [3 _ _] eh inteiro`
`printf("O numero [%f] eh real",X);`	`O numero [2.346] e real`
`printf("O numero [%.2f] eh real", X);`	`O numero [2.35] e real`
`printf("O numero [%.7f] eh real", X);`	`O numero [2.3460000] e real`
`printf("O numero [%10.4f] eh real", X);`	`O numero [_ _ _ _ 2.3460] e real`
`printf("O numero [%-10.4f] eh real", X);`	`O numero [2.3460 _ _ _ _] e real`

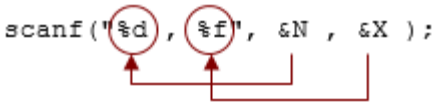
3.7.2. Função scanf()

Essa função corresponde ao leia() do PE. Está definida na biblioteca stdio.h (stdafx.h no VSCE). A forma geral da função *scanf()* é a seguinte:

```
scanf(controle, arg1, arg2, arg3, ...);
```

onde controle é uma cadeia para controle da formatação dos argumentos a serem lidos, e arg1, arg2, arg3, etc, são argumentos que serão lidos conforme definido em controle.

A figura ao lado ilustra como ocorre a substituição dos argumentos na cadeia de controle (N é inteiro, e X é real), onde:



- %d: indica que o valor de uma variável inteira (int) será lido nesta posição: a variável N;
- %f: indica que o valor de uma variável real (float) será lido nesta posição: a variável X.

O comando acima indica que os valores paras as variáveis N e X estão sendo lidas de uma vez só, estando separados por vírgula.

***Nota**. Nunca esquecer de colocar o símbolo **&** na frente das variáveis que estão sendo lidas por scanf(). Isto porque scanf() coloca a entrada convertida nos locais da memória reservados para armazenar o valor das variáveis lidas, e estes locais (endereços) são fornecidos quando colocamos o símbolo & na frente da respectiva variável. Esquecer de colocar este símbolo pode trazer resultados inesperados quando o programa for executado.*

Ao contrário do argumento de controle de *printf()*, aqui não é permitido colocar texto para indicar o que está sendo lido. Para contornar este pequeno problema, faz-se uso de *scanf()* em conjunto com *printf()*, como mostrado no exemplo seguinte.

***Nota**. No VSCE, usar scanf_s() ao invés de scanf(). A forma de usar é a mesma.*

Exemplo 3. Função scanf().

```
   :
int K;
float X;
printf("Forneca o valor de K e de X, separados por virgula: ");
scanf("%d, %f",&K,&X); // Usar scanf_s() no VSCE.
   :
```

Resultado do trecho de programa

```
Forneca o valor de K e de X, separados por virgula: 2,3.4323 [enter]
```

Os números em negrito (vermelho) é o que deverá ser digitado para a leitura por *scanf()*, seguido da tecla [enter] para finalizar o processo de entrada.

3.8. Estruturas de Seleção

3.8.1. Alternativas Simples e Composta - Comando *if*

A forma geral do comando *if* no C/C++ (se no PE) é mostrado na tabela abaixo.

C/C++	Português Estruturado
if (<condição>) <comando-1>;	se (<condição>) então <comando-1>;

if (<condição>)
  <comando-1>;
else
  <comando-2>;
	se (<condição>) então
	  <comando-1>;
	senão
	  <comando-2>;

if (<condição>){
  <comando-1>;
  :
  <comando-n>;
} else {
  <comando-n+1)>;
  :
  <comando-m>;
}
	se (<condição>) então
	  início
	  <comando-1>;
	  :
	  <comando-n>;
	  fim;
	senão
	  início
	  <comando-n+1>;
	  :
	  <comando-m>;
	  fim;

Exemplo 4. Comando *if*.

C/C++	Português Estruturado

if (a > 3) {
  b = a + 2;
  c = b + 2;
}
	se (a > 3) então
	  início
	  b ← a + 2;
	  c ← b + 2;
	  fim;

if (J == 1)
  x = a + 2;
else {
  x = 0;
  z = 0;
}
	se (J = 1) então
	  x ← a + 2;
	senão
	  início
	  x ← 0;
	  z ← 0;
	  fim;

3.8.2. Seleção Múltipla - Comando *switch*

A forma geral do comando *switch* no C/C++ (escolha..fimescolha no PE) é mostrado abaixo (os colchetes, "[" e "]", indicam que a cláusula por eles delimitada é opcional).

C/C++	Português Estruturado

switch(expressão){
  case expressão-constante 1:
  <bloco de comandos-1>
  [break;]
  :
  case expressão-constante n:
  <bloco de comandos-n>
  [break;]
  [default: <bloco de comandos-extra>]
}
	escolha <expressãp>
	  caso expressão-constante 1:
	  <bloco de comandos-1>
	  :
	  caso expressão-constante n:
	  <bloco de comandos-n>
	  senao
	  <bloco de comandos-extra>
	fim escolha;

Este comando avalia a *expressão* e compara o seu resultado com cada *expressão-constante* colocada após a palavra *case*. Caso sejam iguais, executa o *bloco* de comandos que vem na sequência. Caso não haja nenhuma *expressão-constante* igual ao resultado da expressão, então bloco de comandos associado à cláusula *default* (caso ela exista) é executado.

Notar também o uso opcional do comando *break*. Ele existe para interromper a execução do *switch*; isto é, ao ser encontrado o *break*, o fluxo do programa passa para a próxima instrução após o comando *switch*.

Exemplo 5. Comando *switch* (1).

Neste exemplo, todos os comandos do corpo do comando *switch* serão executados se a variável c for igual a 'A': as variáveis capa, lettera e total serão incrementadas. Se c for igual a 'a', lettera e total serão incrementadas, e se c for diferente tanto de 'A' com de 'a', apenas total será incrementada.

Nota. x++ significa x=x+1.

```
switch (c) {
  case 'A':
    capa++;
  case 'a':
    lettera++;
  default: total++;
}
```

Exemplo 6. Comando *switch* (2).

Neste exemplo, se a variável c for igual a 'A', apenas capa será incrementada. Se c for igual a 'a', apenas lettera será incrementada, e se c for diferente tanto de 'A' como de 'a', apenas total será incrementada. Isto porque o comando *break* faz com que o comando *switch* termine naquele ponto. Ou seja, o comando *break* colocado ao final da opção *case* correspondente garante que apenas os comandos associados àquela opção serão executados.

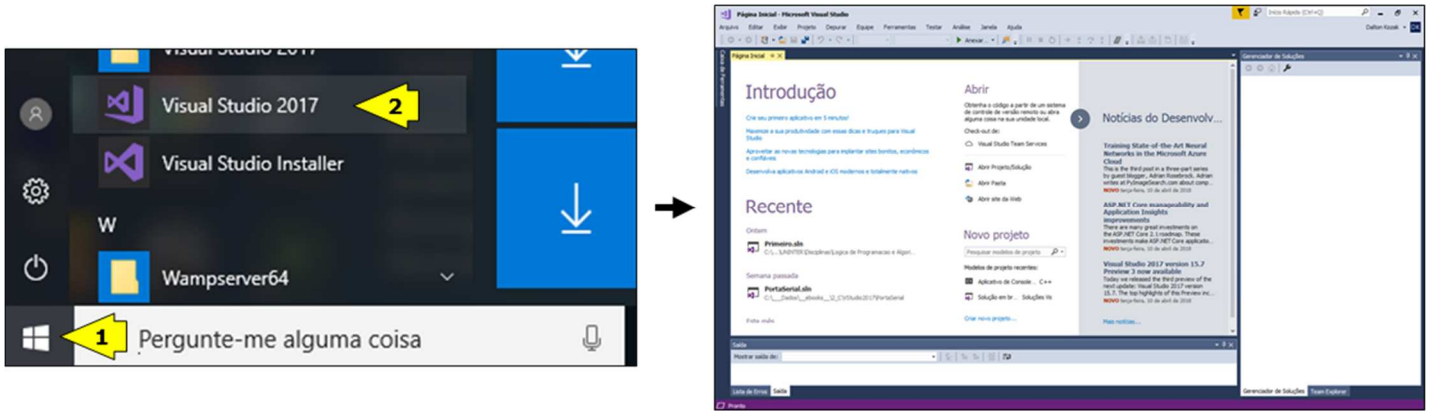
```
switch (c) {
  case 'A':
    capa++;
    break;
  case 'a':
    lettera++;
    break;
  default: total++;
}
```

4. Seqüência da Prática

Abaixo está descrito como criar e rodas programas nas duas IDEs: VSCE e Dev-C++. Escolha a que melhor convier.

4.1. Criando um Programa (Projeto) no VSCE

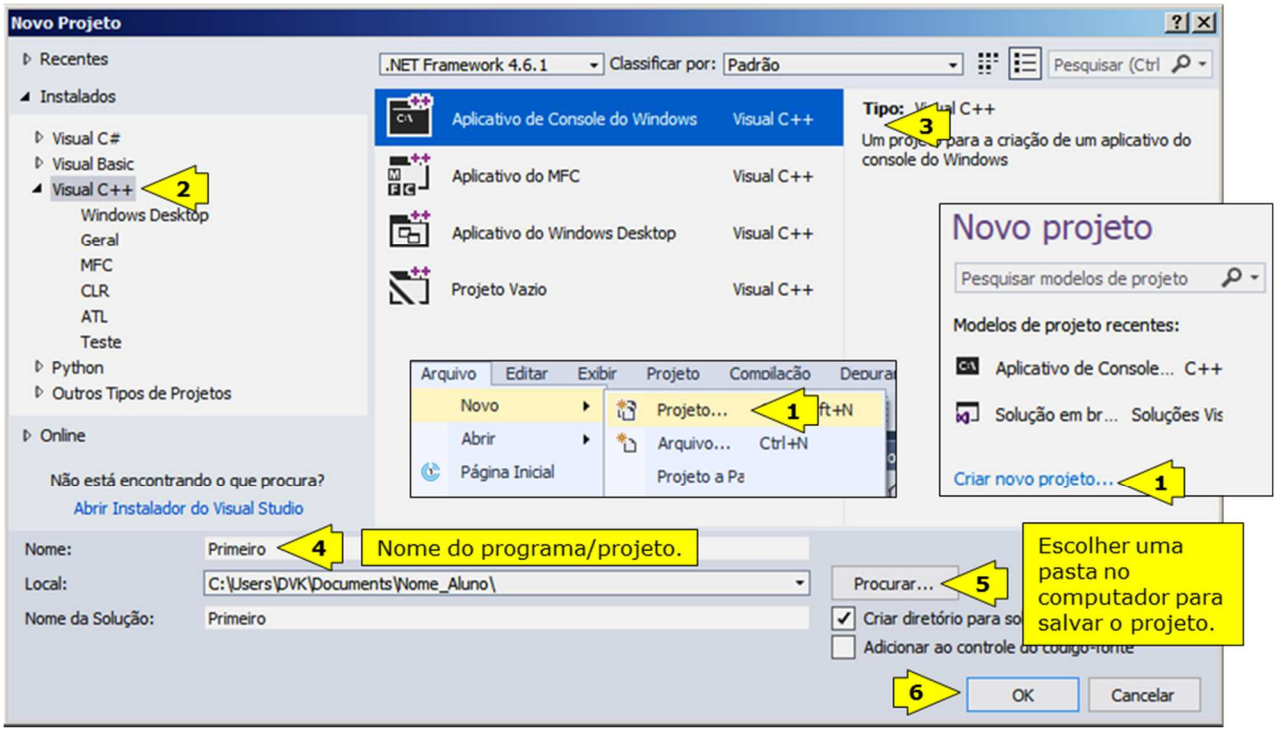
Primeiro passo é abrir o VSCE conforme ilustrado abaixo. Se for o primeiro acesso (que pode demorar), pode ser perguntado se é desejado entrar numa conta; neste caso, responder negativamente.



A seqüência para criar o programa (na forma de um projeto em C/C++) uma vez aberto o VSCE é a seguinte:

1. clicar em Criar novo projeto..., ou em *Arquivo/Novo/Projeto...* se o VSCE já estiver aberto: uma janela para criação de um novo projeto será aberta;
2. selecionar Visual C++, caso não esteja selecionado;
3. selecionar Aplicativo de Console do Windows Visual C++;
4. digitar o nome do programa (por exemplo, "Primeiro");
5. procurar um local (pasta) para armazenar o novo projeto;
6. clicar em [OK].

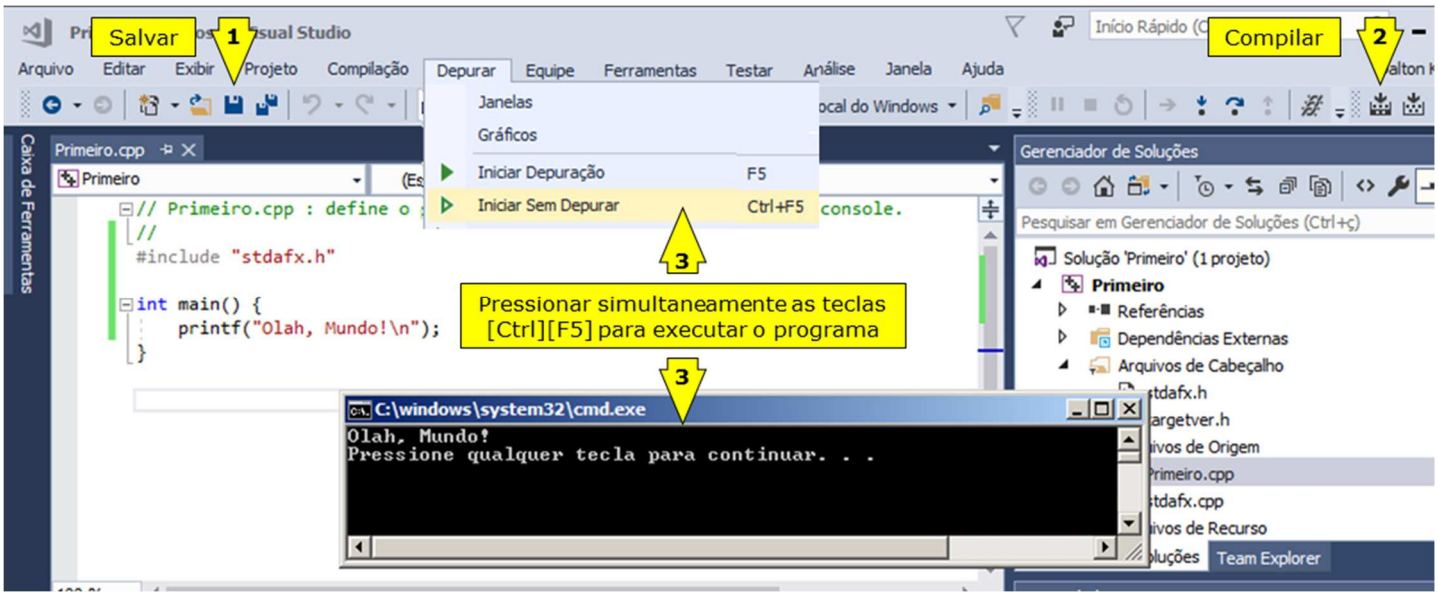
Essa seqüência está ilustrada na figura abaixo.



Realizado o procedimento acima, será aberta uma tela com o esqueleto do programa mínimo. Altere-o para o abaixo

```
int main(){
    printf("Olah, Mundo!\n");
}
```

Após salvá-lo, o próximo passo será compilá-lo e executá-lo conforme ilustrado na figura abaixo. Esse programa gera a seguinte mensagem: "Olah, Mundo!". Apesar de bem simples, será esta a seqüência a ser sempre realizada para executar qualquer programa.



4.2. Criando um Programa no Dev-C++

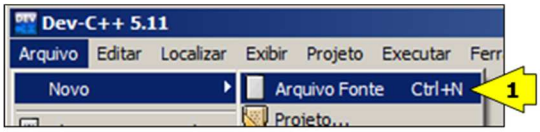
Inicialmente, procurar na lista de programas do Windows por *Bloodshed Dev-C++/Dev-C++* e abrir este programa.

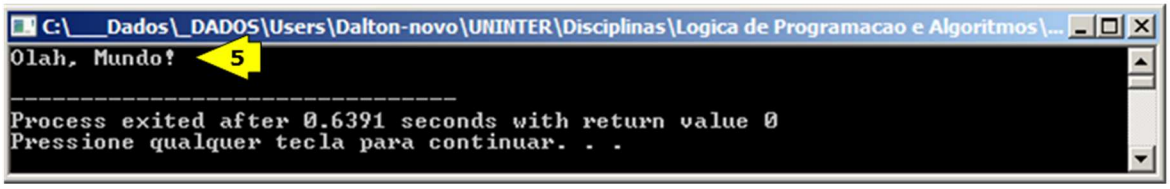
Uma vez aberto o programa, a seqüência (ver figura abaixo) para criar e rodar um programa no Dev-C++ é a seguinte:

7. Criar um arquivo fonte novo clicando em *Arquivo/Novo/Arquivo Fonte.*
8. Digitar o seguinte programa (o que está depois de //, inclusive, não precisa digitar: são comentários apenas):

```
#include "stdio.h" // Para usar printf().
int main(){
    printf("Olah, Mundo!\n"); // Função de impressão na tela.
}
```

9. Salvar o arquivo do programa (primeiro.cpp, por exemplo) em uma pasta conhecida.
10. Clicar no ícone para compilar/executar (ver figura abaixo), ou pressionar [F11].
11. Se o programa estiver correto (sem erros de sintaxe) a tela de saída será apresentada conforme mostra a figura,





4.3. Executando Programas Básicos

4.3.1. Somando dois Números

Implemente o programa abaixo em C, e o teste com várias entradas.

C/C++	Português Estruturado

`#include "stdafx.h" // "stdio.h" no C padrão.`
`int main() {`
`  float a, b;`
`  printf("Digite a e b: ");`
`  // scanf_s() é equivalente a scanf().`
`  scanf_s("%f %f", &a, &b);`
`  printf("a + b = %.2f\n", a + b);`
`}`
	`inicio`
	`  real: a,b;`
	`  imprimal("Digite a e b:");`
	`  leia(a,b);`
	`  imprimal("a+b=",a+b);`
	`fim.`

4.3.2. Verificando se um Número Inteiro é Par ou ímpar

Implemente o programa abaixo em C, e o teste com várias entradas.

C/C++	Português Estruturado

`#include "stdafx.h" // "stdio.h" no C padrão.`
`int main() {`
`  int n;`
`  printf("Digite n: ");`
`  scanf_s("%d", &n);`
`  if (n%2==0)`
`  printf("%d eh par.\n", n);`
`  else`
`  printf("%d eh impar.\n", n);`
`}`
	`inicio`
	`  inteiro: n;`
	`  imprimal("Digite n:");`
	`  leia(n);`
	`  se (n mod 2=0) entao`
	`  imprimal(n," eh par.");`
	`  senão`
	`  imprimal(n," eh impar.");`
	`fim.`

4.3.3. Verificando Códigos de Cores

Conforme um código, uma cor deve ser impressa. Os códigos são:

1: "Azul"; 2: "Verde"; 3: "Roxo"; outros: "Branco".

Implemente o programa abaixo em C, e o teste com várias entradas.

C/C++	Português Estruturado

`#include "stdafx.h" // "stdio.h" no C padrão.`
`int main() {`
`  int c;`
`  printf("Digite o codigo da cor: ");`
`  scanf_s("%d", &c);`
`  switch (c) {`
`  case 1: printf("%d: Azul\n", c); break;`
`  case 2: printf("%d: Verde\n", c); break;`
`  case 3: printf("%d: Roxo\n", c); break;`
`  default: printf("%d: Branco\n", c); break;`
`  }`
`}`
	`inicio`
	`  inteiro: c;`
	`  imprimal("Digite o codigo:");`
	`  leia(c);`
	`  escolha(c)`
	`  caso 1: imprimal(c,": Azul.");`
	`  caso 2: imprimal(c,": Verde.");`
	`  caso 3: imprimal(c,": Roxo.");`
	`  senao  imprimal(c,": Branco.");`
	`  fimescolha;`
	`fim.`

5. Exercícios

5.1. Determinando o Maior Número

Implemente um programa em C que, dados três números quaisquer, imprima o maior deles.

5.2. Verificando Código de Cores

Conforme um código, uma disciplina deve ser impressa. Os códigos são:

1: "Matemática"; 2: "Física"; 3: "Química"; 4: "Português"; outros: "Não definido.".

Implemente o programa em C que realize isso.