

Estrutura de Dados 1

Aula 08 – TAD Tipo Abstrato de Dados

Antonio Angelo de Souza Tartaglia

angelot@ifsp.edu.br

Estrutura de Dados 1

Criando Bibliotecas em C

- ▶ Quando colocamos em nossos programas as chamadas para bibliotecas externas, por exemplo `#include <stdio.h>`, na realidade é como se estivéssemos transcrevendo todo o código que tal biblioteca contém, para o código que estamos desenvolvendo;
- ▶ Imagine se todos esses códigos aparecessem quando estamos codificando um programa. Um simples programa que gera um “hello world” teria centenas de linhas de código;
- ▶ A utilidade dessas bibliotecas ou **headers**, é a de criar um arquivo que contém diversas funções específicas, separadas e organizadas por assunto.

Estrutura de Dados 1

Criando Bibliotecas em C

- Para criação de um exemplo utilizaremos um exercício anterior:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void func_it(int n);
5  int func_rec(int n);
6
7  int main() {
8      int n, j;
9      int res_it, res_rec;
10     printf("Entre com numero: ");
11     scanf("%d", &n);
12     printf("Versao iterativa\n");
13     func_it(n);
14     printf("\n\n");
15     printf("Versao Versao recursiva\n");
16     printf("\n\n");
17     j = func_rec(n);
18     printf("\n\n\n\n");
19     system ("pause");
20
21 }
```

```
23 void func_it(int n) {    //função iterativa
24     n++;
25     do{
26         n--;
27         printf("\n%d", n);
28     }while(n);
29 }
30
31 int func_rec(int n) {    //função recursiva
32     printf("\n%d", n);
33     if (n == 0){
34         return 0;
35     }
36     return (func_rec(n-1));
37 }
```

Colocaremos estas
funções em uma
bibilioteca

Criando Bibliotecas em C

- Simplesmente retiramos as funções junto com seus protótipos e as colocamos em outro arquivo que salvaremos como “**minhalib.h**” na mesma pasta do projeto original e acrescentamos mais 1 include, o de nossa nova biblioteca “**minhalib.h**”, no arquivo principal:

```
1 #include <stdio.h>    //arquivo original .c
2 #include <stdlib.h>
3 #include "minhalib.h"
4
5 int main(){
6     int n, j;
7     int res_it, res_rec;
8     printf("Entre com numero: ");
9     scanf("%d", &n);
10    printf("Versao iterativa\n");
11    func_it(n);
12    printf("\n\n");
13    printf("Versao Versao recursiva\n");
14    printf("\n\n");
15    j = func_rec(n);
16    printf("\n\n\n\n");
17    system ("pause");
18 }
```

```
1 //arquivo "minhalib.h"
2
3 void func_it(int n);
4 int func_rec(int n);
5
6 void func_it(int n){    //função iterativa
7     n++;
8     do{
9         n--;
10        printf("\n%d", n);
11    }while(n);
12 }
13 int func_rec(int n){    //função recursiva
14     printf("\n%d", n);
15     if (n == 0){
16         return 0;
17     }
18     return (func_rec(n-1));
19 }
```

Estrutura de Dados 1

Criando Bibliotecas em C

- ▶ Se você abrir um Header, os arquivos .h que costumamos usar, verá que eles contêm diversos protótipos de funções, muitos comentários e outras coisas. Mas código C, você verá muito pouco;
- ▶ Nos **Headers**, são declaradas todas as funções, bem como comentários a respeito delas, explicando o que são, para que servem, que parâmetros recebem e o que retornam;
- ▶ Os arquivos .h, são uma espécie de intermediário, entre seu programa e os módulos. É nos arquivos “**Headers**” o local onde é explicado o que cada função que foi definida nos módulos faz, ou seja, o “**Header**” é uma **interface de comunicação**.

Estrutura de Dados 1

Dividindo o programa em módulos

- ▶ Apenas lendo o Header temos condições de saber como utilizar tudo que o módulo tem a oferecer;
- ▶ Mas o mais importante: não precisamos saber de qual maneira as funções que pertencem ao módulo foram implementadas. **Não precisamos conhecer o seu código;**
- ▶ Ao contrário de ler um módulo com centenas de linhas de código, teremos contato apenas com o “**Header**”, um arquivo bem pequeno, simples e explicativo.

Estrutura de Dados 1

Dividindo o programa em módulos

- ▶ Até agora nossos programas têm sido pequenos e simples e com uma única “folha” de código. Imagine porém, se você estivesse programando um jogo em C, com módulos das funções de áudio, imagem, fases, personagens, jogabilidade, etc. Seu módulo principal (main.c) ficaria impossível de se trabalhar, tamanho o número de protótipos. É aí que entram os arquivos **Headers** ou simplesmente .h;
- ▶ Colocamos neles, todos os protótipos, e ainda mais: colocaremos descrições simples diretas e completas. Essa é a parte importante, porque está neste arquivo que o utilizador da biblioteca terá acesso às informações de como utilizar as funções disponíveis na biblioteca.

Estrutura de Dados 1

Dividindo o programa em módulos

- ▶ Utilizando nosso exemplo anterior, onde criamos uma biblioteca com os protótipos e as funções e em seguida referenciamos esta biblioteca no programa principal, faremos mais uma mudança:
- ▶ Criaremos mais um arquivo de código com extensão .c . Este arquivo deve ter **obrigatoriamente** o mesmo nome do arquivo .h. Ele conterá as funções propriamente ditas;
- ▶ No arquivo .h deixaremos **somente os protótipos das funções** e colocaremos **comentários de como funcionam**, o que recebem como parâmetro, o que retornam, etc. Utilizaremos descrições detalhadas.

Estrutura de Dados 1

Criando projetos – CodeBlocks

- ▶ Mas para que funcione, é necessária uma mudança na utilização da **IDE CodeBlocks**.
- ▶ Agora é necessária a criação de um projeto no ambiente de programação.
- ▶ Como são módulos separados, são compilados separadamente, portanto precisamos que sejam unidos durante a compilação. Inicializando um projeto dentro da IDE conseguimos isso, todos os arquivos que fazem parte do projeto serão unidos durante a fase de compilação.

IDE: Integrated Development Environment, ou
Ambiente de Desenvolvimento Integrado

Estrutura de Dados 1

Criando projetos – CodeBlocks

- ▶ Antes das IDE's utilizávamos linhas de comando para essa ligação com programas específicos. Chamávamos estas operações de *linkedição*, ou *lincagem*.
- ▶ Podemos, usando nosso exemplo, compilar os dois arquivos codificados separadamente, o main.c e o minhalib.c, em um arquivo executável em uma única linha de comando:

```
gcc -o prog1.exe main.c minhalib.c
```

- ▶ Porém, normalmente compilamos cada código separadamente em um arquivo **objeto**, e os unimos (*linkedição*), em uma fase posterior. Neste caso mudanças efetuadas em um dos arquivos, dispensa a recompilação dos outros que não foram alterados:

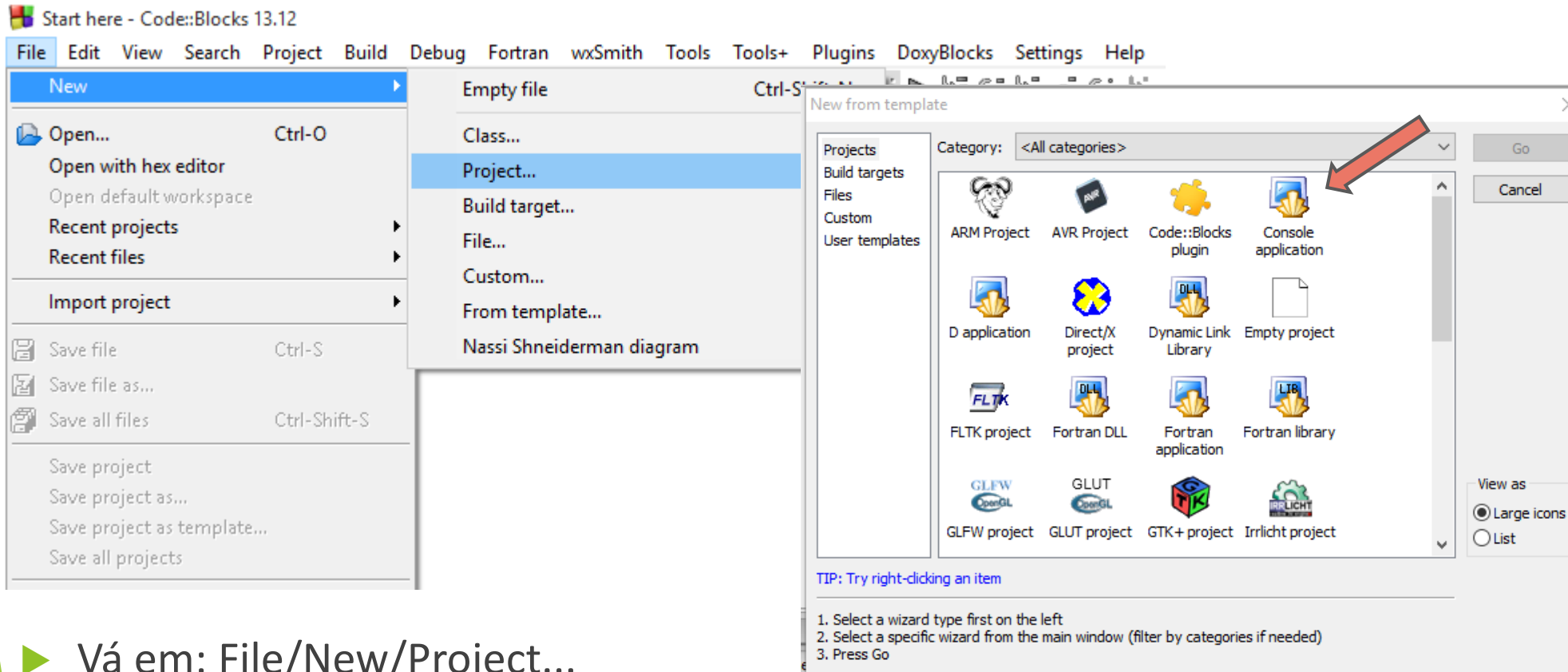
```
gcc -c main.c  
gcc -c minhalib.c  
gcc -o prog1.exe main.o minhalib.o
```

-o: especifica o nome do arquivo de saída
-c: compila o código fonte em um arquivo objeto com extensão .o

Estrutura de Dados 1

Criando projetos – CodeBlocks

- Compilando agora, utilizando a IDE CodeBlocks:

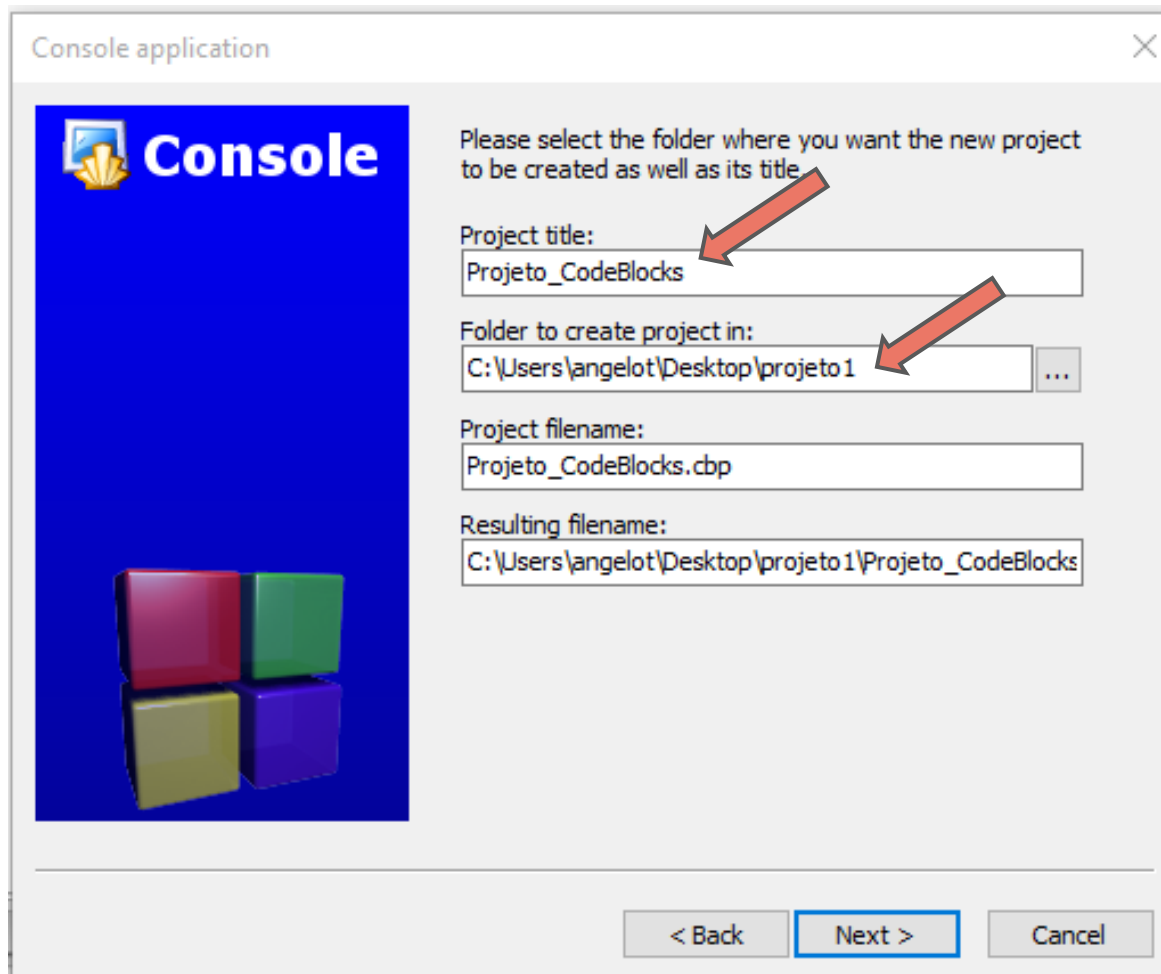


- Vá em: File/New/Project...
- Na nova janela, escolha: *Console application*. Na próxima janela *Next*, escolha “C”, dê um nome ao projeto, escolha a pasta para criação e salve o projeto.

Estrutura de Dados 1

Criando projetos – CodeBlocks

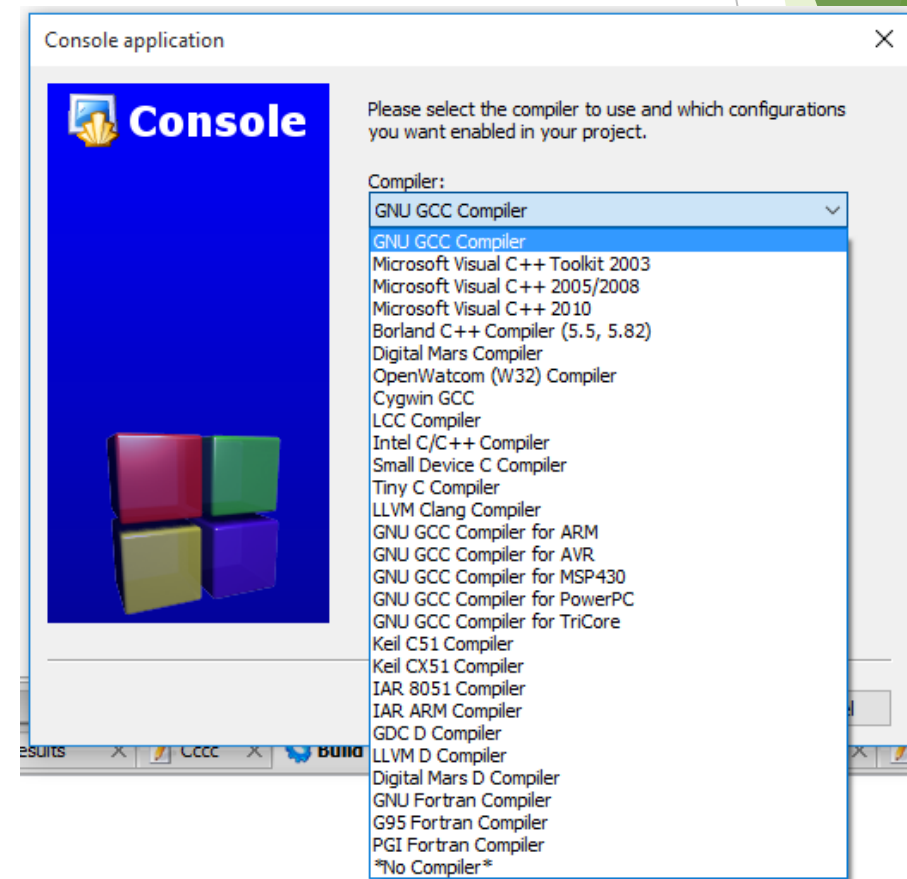
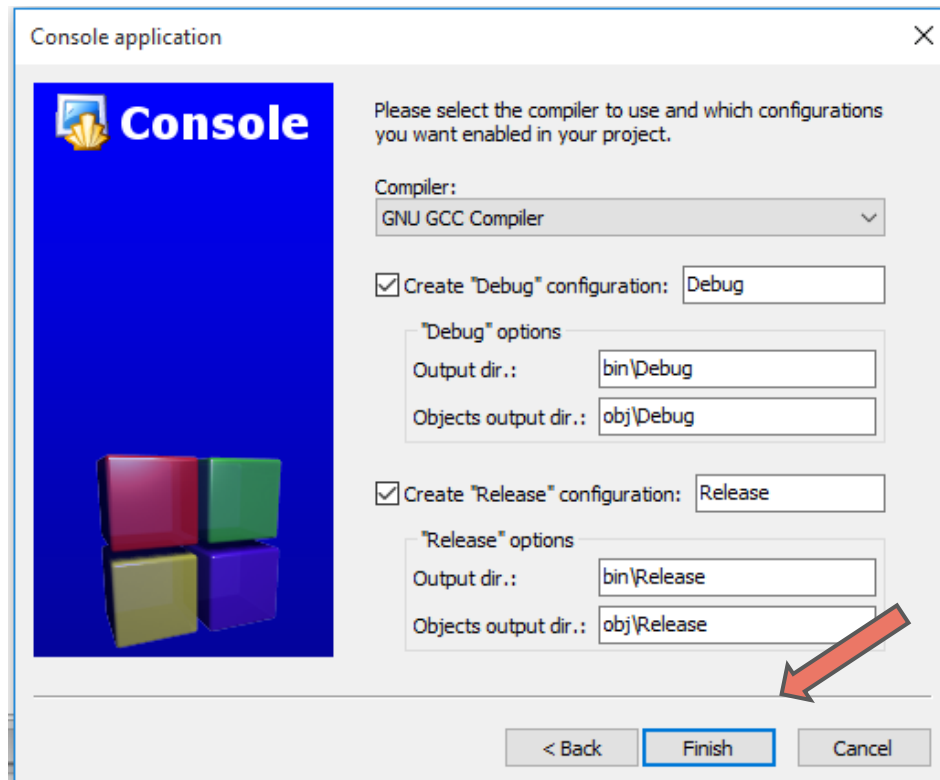
- Dê um nome ao seu novo projeto e escolha onde será salvo:



Estrutura de Dados 1

Criando projetos – CodeBlocks

- ▶ Em seguida o CodeBlocks exibe um resumo das configurações de criação do projeto. Neste ponto é possível alterar algumas configurações, como por exemplo o compilador:

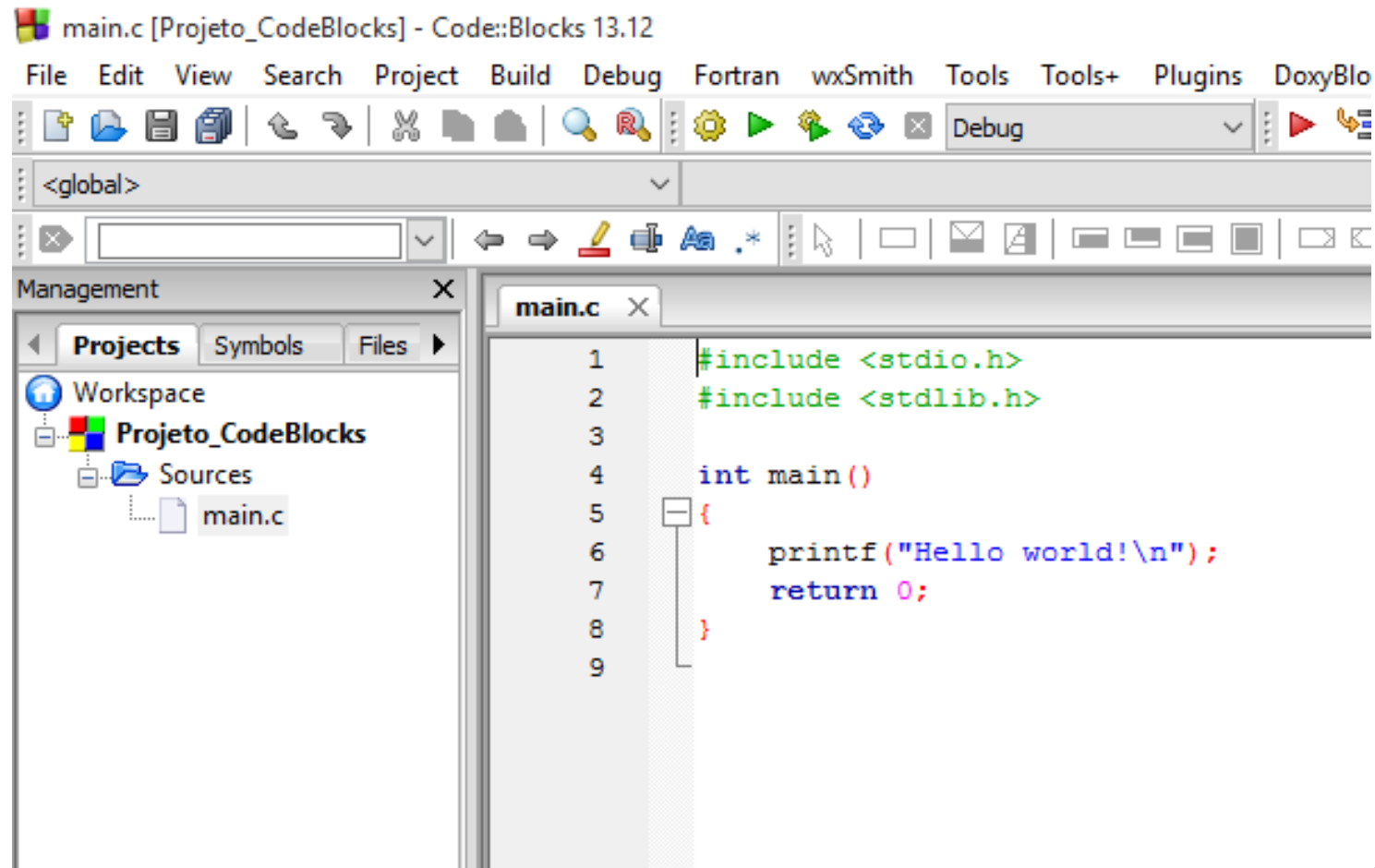


- ▶ Não sendo necessária nenhuma alteração, clique em Finish.

Estrutura de Dados 1

Criando projetos – CodeBlocks

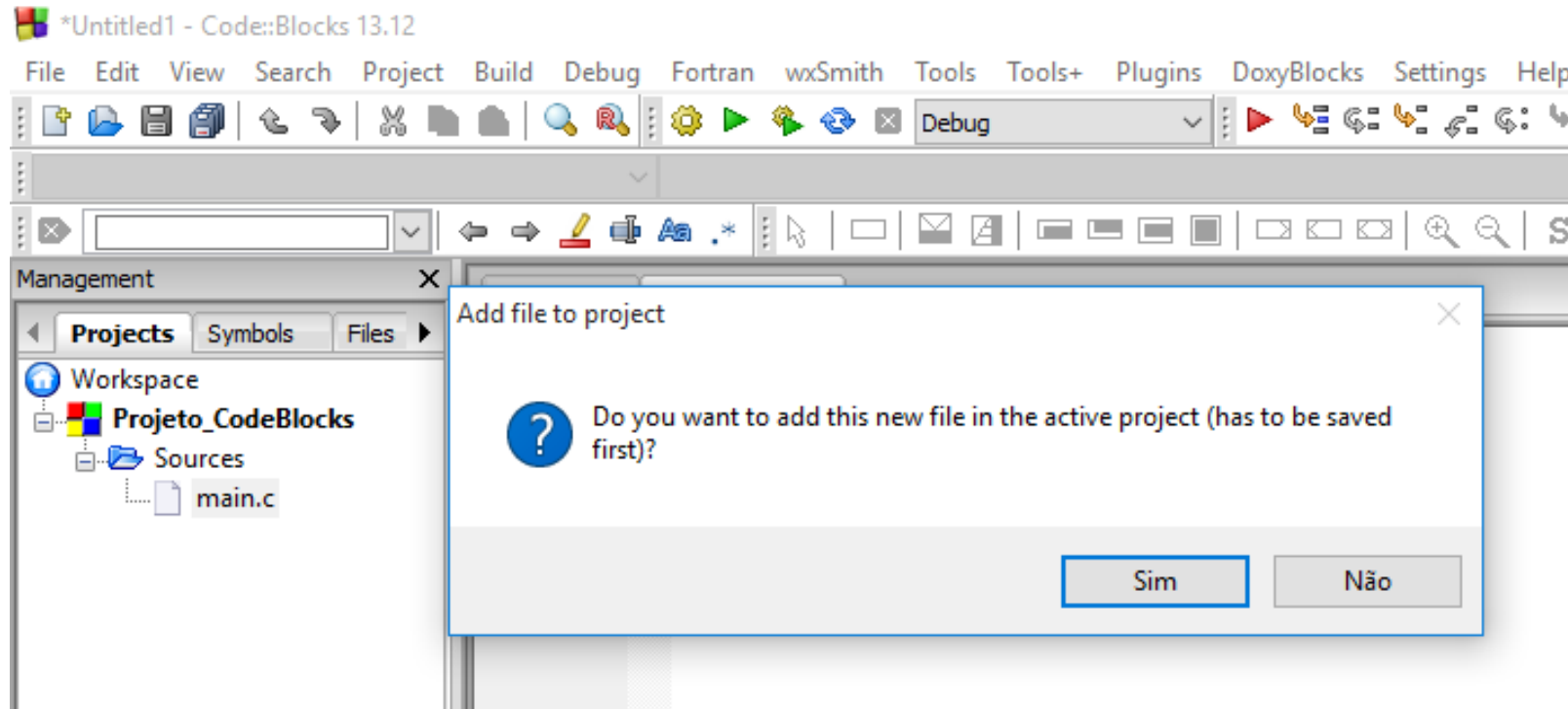
- Com o projeto criado, teremos esse resultado:



Estrutura de Dados 1

Criando projetos – CodeBlocks

- ▶ Agora é só criar os arquivos e codificá-los. Adicione um novo arquivo, faça isso normalmente no menu *File/New/Empty File*, em seguida CodeBlocks irá perguntar se você quer adicioná-lo ao projeto atual, responda que sim:

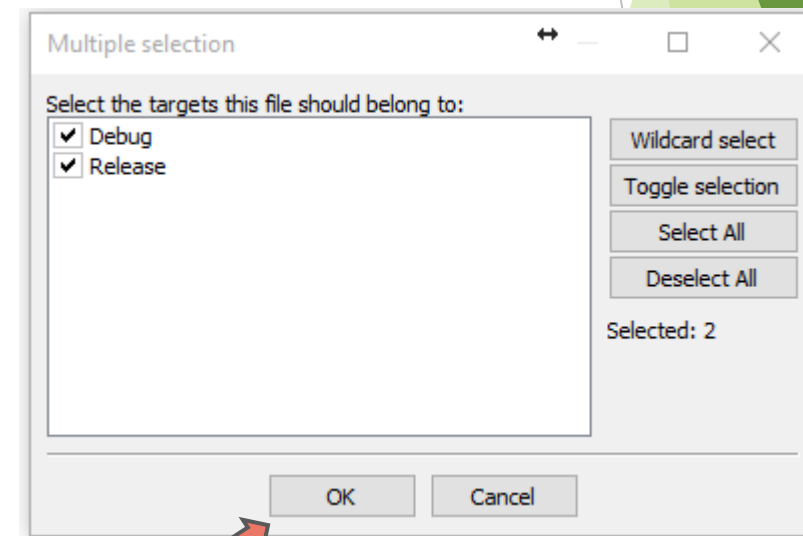
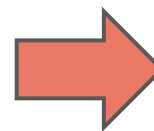
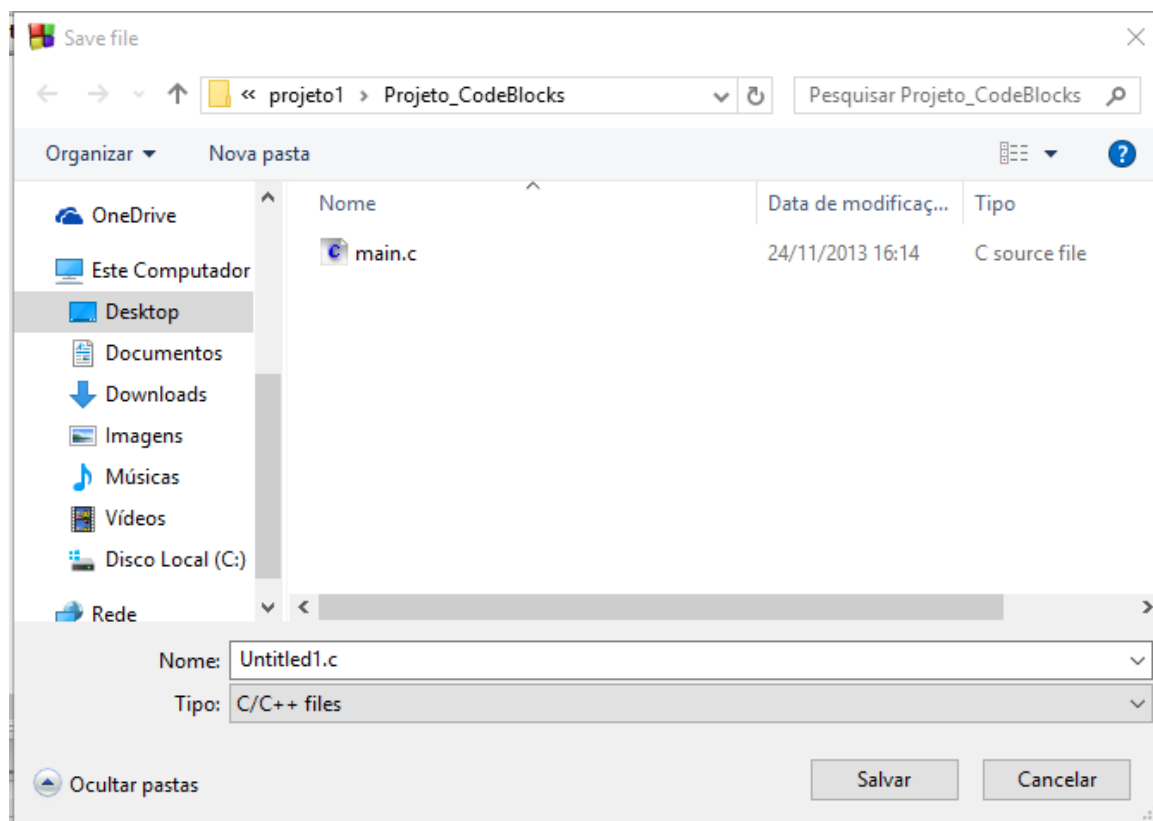


- ▶ Criaremos 2 arquivos para nosso projeto: “minhalib.h” e “minhalib.c”.

Estrutura de Dados 1

Criando projetos – CodeBlocks

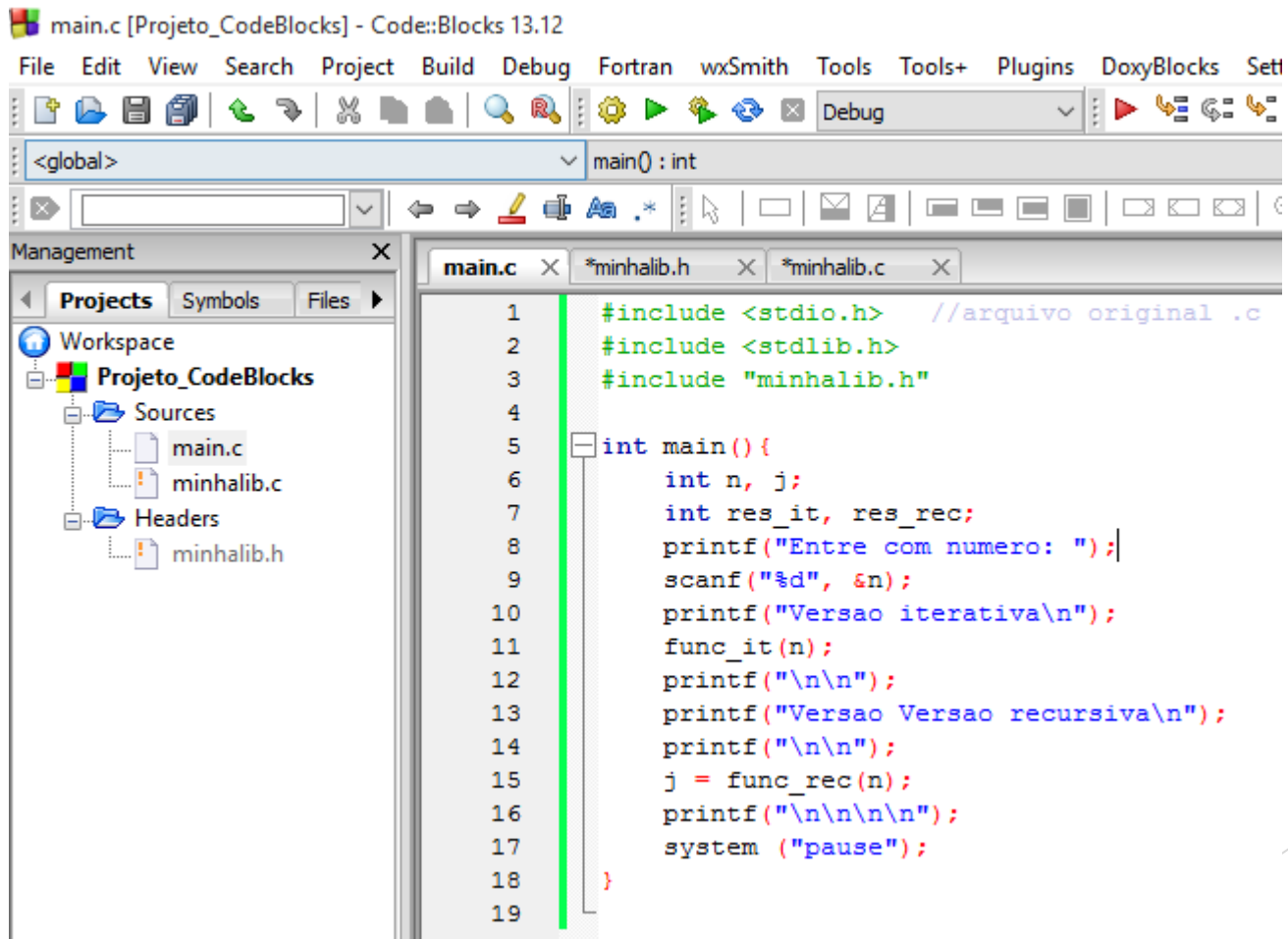
- ▶ Abrirá então uma janela padrão do sistema já direcionada para a pasta onde você criou o seu projeto, para que o arquivo seja salvo. Nomeie-o “minhalib.h” e salve-o. Uma caixa de diálogo aparecerá, clique em ok. Em seguida repita toda a operação de criação para um segundo arquivo, e salve-o com o nome de “minhalib.c”:



Estrutura de Dados 1

Criando projetos – CodeBlocks

- ▶ Agora com os arquivos abertos na IDE, é só digitar os códigos;
- ▶ O programa principal:



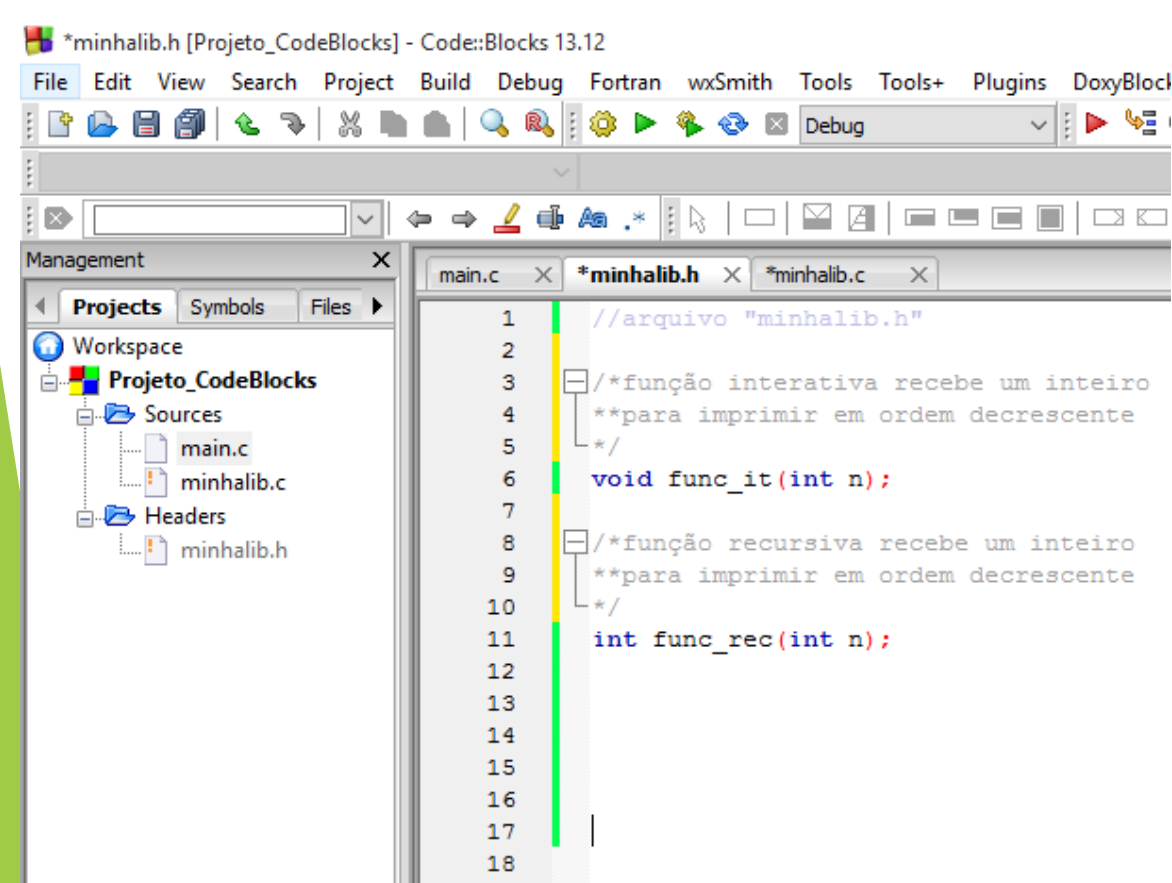
main.c [Projeto_CodeBlocks] - Code::Blocks 13.12

```
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Seti
<global> main() : int
Management
Projects Symbols Files
Workspace
Projeto_CodeBlocks
Sources
main.c
minhalib.c
Headers
minhalib.h
1 #include <stdio.h> //arquivo original .c
2 #include <stdlib.h>
3 #include "minhalib.h"
4
5 int main(){
6     int n, j;
7     int res_it, res_rec;
8     printf("Entre com numero: ");
9     scanf("%d", &n);
10    printf("Versao iterativa\n");
11    func_it(n);
12    printf("\n\n");
13    printf("Versao Versao recursiva\n");
14    printf("\n\n");
15    j = func_rec(n);
16    printf("\n\n\n\n");
17    system ("pause");
18 }
19
```

Estrutura de Dados 1

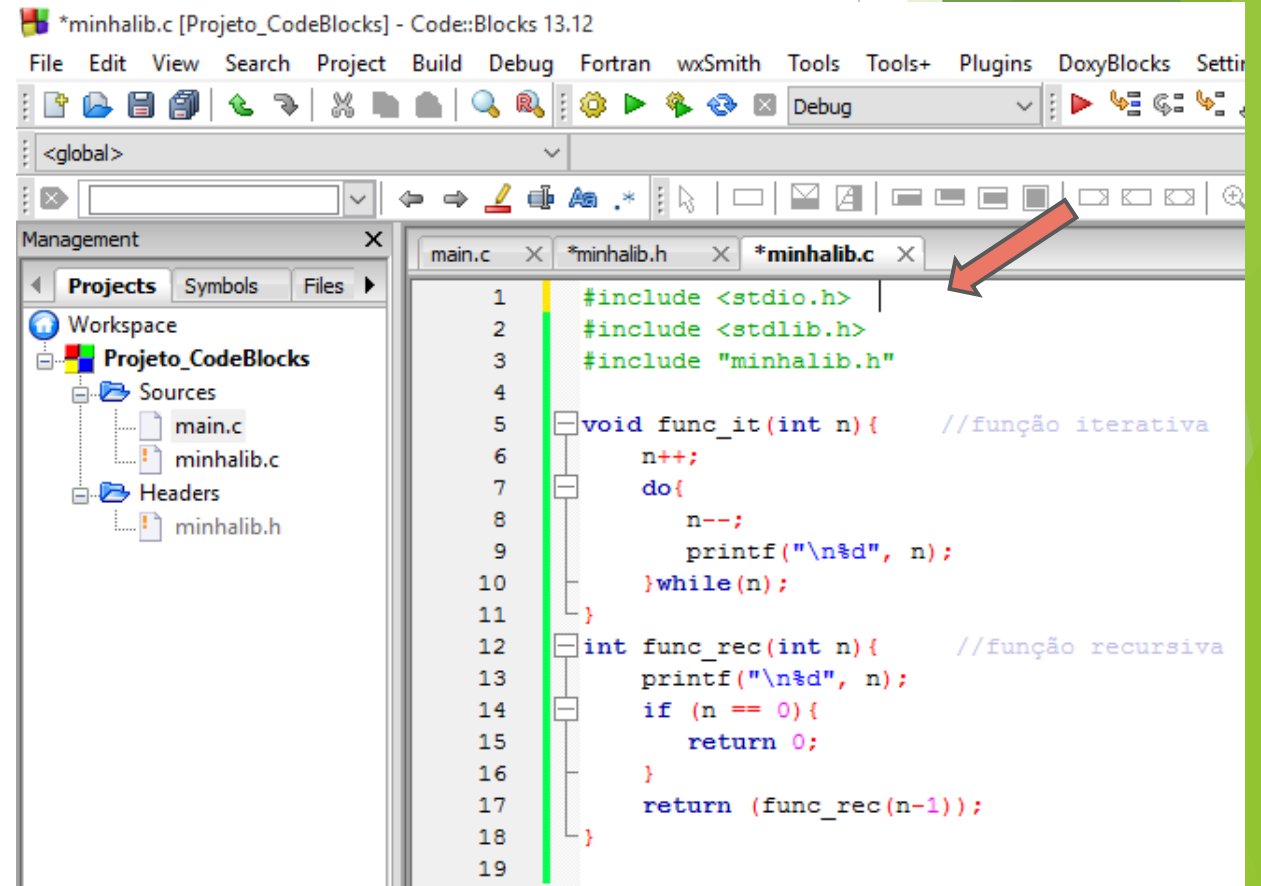
Criando projetos – CodeBlocks

- ▶ Arquivo Header minhalib.h e o módulo das funções “minhalib.c”, que **também deve ter as bibliotecas referenciadas**:



The screenshot shows the Code::Blocks 13.12 IDE with the project "Projeto_CodeBlocks" open. The "minhalib.h" header file is selected in the editor. The file contains two function declarations: a recursive function "func_rec" and an iterative function "func_it". The "Management" panel on the left shows the project structure with "main.c", "minhalib.c", and "minhalib.h" in the "Sources" folder.

```
1 //arquivo "minhalib.h"
2
3 /*função interativa recebe um inteiro
4 **para imprimir em ordem decrescente
5 */
6 void func_it(int n);
7
8 /*função recursiva recebe um inteiro
9 **para imprimir em ordem decrescente
10 */
11 int func_rec(int n);
12
13
14
15
16
17
18
```



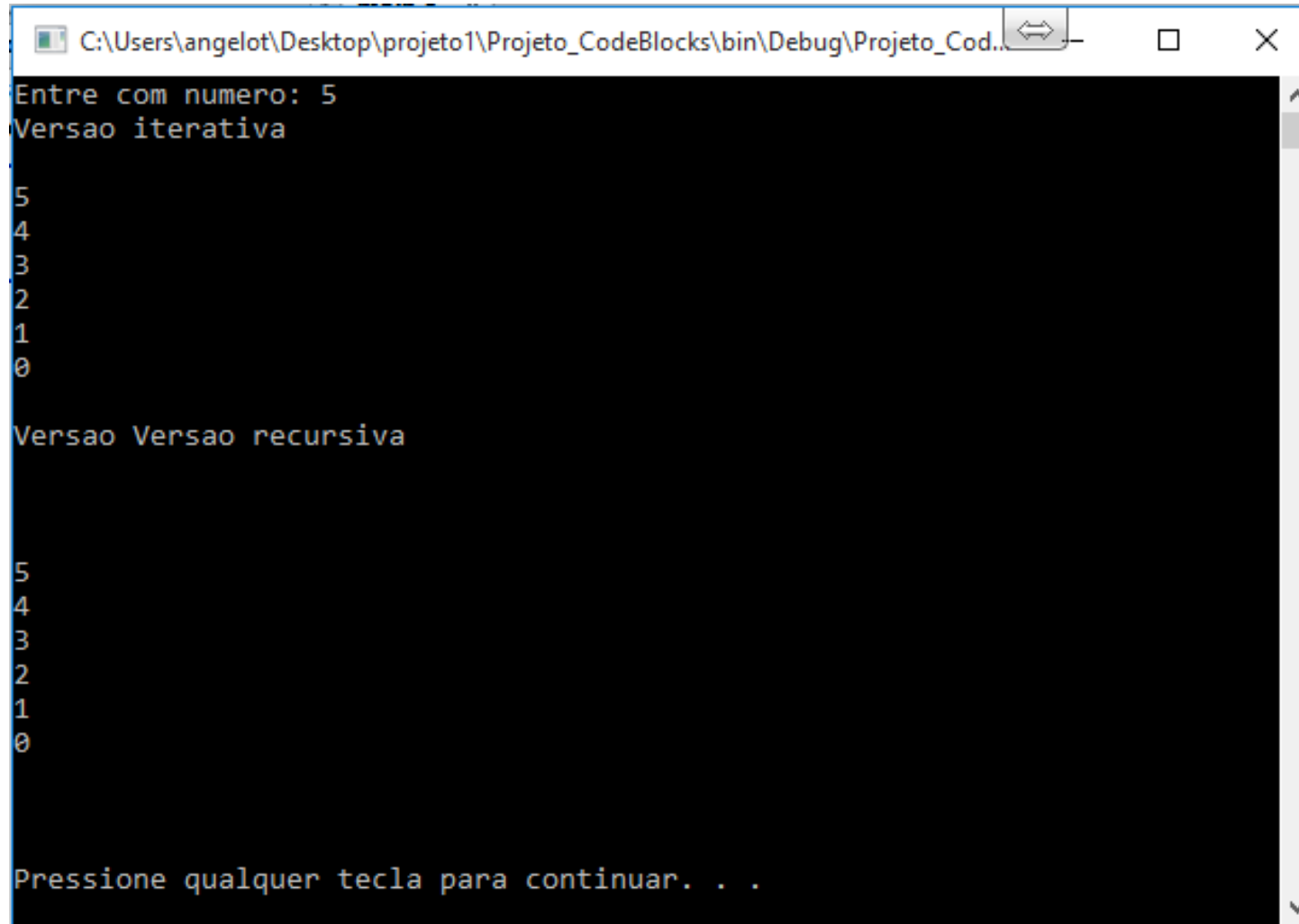
The screenshot shows the Code::Blocks 13.12 IDE with the project "Projeto_CodeBlocks" open. The "minhalib.c" source file is selected in the editor. The file includes "stdio.h", "stdlib.h", and "minhalib.h". It implements the "func_it" and "func_rec" functions. A red arrow points to the "minhalib.c" tab in the editor. The "Management" panel on the left shows the project structure with "main.c", "minhalib.c", and "minhalib.h" in the "Sources" folder.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "minhalib.h"
4
5 void func_it(int n){ //função interativa
6     n++;
7     do{
8         n--;
9         printf("\n%d", n);
10    }while(n);
11 }
12 int func_rec(int n){ //função recursiva
13     printf("\n%d", n);
14     if (n == 0){
15         return 0;
16     }
17     return (func_rec(n-1));
18 }
19
```

Estrutura de Dados 1

Criando projetos – CodeBlocks

- Agora é só compilar, e o resultado:



```
C:\Users\angelot\Desktop\projeto1\Projeto_CodeBlocks\bin\Debug\Projeto_Cod...
Entre com numero: 5
Versao iterativa
5
4
3
2
1
0

Versao Versao recursiva

5
4
3
2
1
0

Pressione qualquer tecla para continuar. . .
```

Estrutura de Dados 1

Atividade 1

- ▶ Reescreva o programa da 2ª aula, atividade 02 cálculo de vantagens e cálculo de deduções, criando um arquivo Header (biblioteca) como interface, e um módulo que conterá as funções (arquivo .c);
- ▶ Entregue no Moodle como atividade 1.

Estrutura de Dados 1

Tipo Abstrato de Dado

► Definição de tipo de dado:

- Conjunto de valores que uma variável pode assumir, por exemplo para o tipo int:

-n... -2, -1, 0, 1, 2, ...n

► Definição de Estrutura de dados:

- Conjunto de variáveis que possuem um relacionamento lógico entre tipos de dados exemplo:

```
struct funcionario{  
    int    id;  
    char   nome[30];  
    char   departamento[15];  
    float  salario;  
};
```

Nesta estrutura, as variáveis que a compõe passam a ter um significado

Estrutura de Dados 1

Tipo Abstrato de Dado

- ▶ Tipo Abstrato de Dado ou TAD, pode ser visto como um modelo matemático;
- ▶ TADs incluem, além da estrutura lógica dos dados, operações para manipulação desses dados. Essas operações são a única forma de acesso aos dados;
- ▶ Diferente da estrutura, não temos acesso direto ao que está dentro do TAD. Só acessamos seus dados por meio das operações, ou funções, que trabalham com este tipo abstrato:
 - Criação da estrutura;
 - Inclusão de um elemento;
 - Remoção de um elemento;
 - Acesso a um elemento;
 - Etc.
- ▶ Ideia do TAD: criar funções que vão interagir com os seus dados, e estes dados ficam ocultos do programador/usuário.

Estrutura de Dados 1

Tipo Abstrato de Dado

- ▶ Com o TAD, existe uma separação entre a definição conceitual e a implementação:
 - O programador não tem acesso à implementação;
 - O programa principal acessa o TAD por meio de suas operações, **nunca diretamente**.

- ▶ Reuso;
 - O TAD pode ser acessado por diferentes programas.

- ▶ O TAD é compilado separadamente.

Tipo Abstrato de Dado

► Vantagens:

- Encapsulamento e segurança: usuário/programador não tem acesso direto aos dados;
- Flexibilidade e reutilização: podemos alterar o TAD sem alterar as aplicações.

► Não importa, para o usuário/programador, como foi implementado o TAD, a implementação é separada da aplicação;

► Como exemplo de TAD, criaremos um ponto definido por suas coordenadas “x” e “y”

```
//dados do TAD  
//para guardar a info
```

```
struct ponto{  
    float x;  
    float y;  
}
```

1º passo:

- Definir o arquivo “.h”;
- Protótipos das funções;
- Tipos de ponteiros;
- Dados globalmente acessíveis.

Estrutura de Dados 1

Tipo Abstrato de Dado

- ▶ A convenção em Linguagem C é preparar dois arquivos para implementar um TAD. TADS são criados em módulos.
 - Arquivo “.h” – Contém os protótipos das funções, **tipos de ponteiro**, e dados que são globalmente acessíveis (por exemplo as constantes);
 - Arquivo “.c” – Contém a declaração **do tipo de dado** e implementação de suas funções de acesso.
- ▶ Assim separamos o “conceito” (definição de tipo) de sua implementação

No arquivo “.h” é definido o ponteiro, mas o tipo do dado está no arquivo “.c”, desse modo os dados ficam ocultos do usuário/programador. Este, só consegue acessá-los por meio das funções implementadas no TAD .

Tipo Abstrato de Dado

- ▶ Como exemplo de implementação completa de um TAD, vamos considerar a criação de um tipo de dado para representar um ponto no plano cartesiano. Para isso vamos definir um tipo abstrato, que chamaremos de Ponto, e o conjunto de funções que operam sobre este tipo;
- ▶ Neste exemplo, vamos considerar as seguintes operações:
 - cria: operação que cria um ponto com coordenadas x e y ;
 - libera: operação que libera a memória alocada por um ponto;
 - acessa: operação que devolve as coordenadas de um ponto;
 - atribui: operação que atribui novos valores às coordenadas de um ponto;
 - distancia: operação que calcula a distância entre dois pontos.

Estrutura de Dados 1

Tipo Abstrato de Dado

- A interface (arquivo “ponto.h”) será dada pelo código:

```
1 //Arquivo Ponto.h
2 //Atribui novo nome para struct ponto: Ponto
3 typedef struct ponto Ponto;
4
5 //Cria um novo ponto - somente ponteiro!!
6 Ponto *pto_cria(float x, float y);
7
8 //Libera um ponto
9 void pto_libera(Ponto *p);
10
11 //Acessa valores "x" e "y" de um ponto
12 void pto_acessa(Ponto *p, float *x, float *y);
13
14 //Atribui os valores "x" e "y" a um ponto
15 void pto_atribui(Ponto *p, float x, float y);
16
17 //Calcula a distância entre dois pontos
18 float pto_distancia(Ponto *p1, Ponto *p2);
```

Note que a composição da estrutura Ponto (struct ponto) não é exportada pelo módulo. Dessa forma os demais módulos que usem este TAD não poderão acessar diretamente os campos dessa estrutura. Os clientes desse TAD só terão acesso às informações que possam ser obtidas através das funções exportadas pelo arquivo ponto.h

Tipo Abstrato de Dado

- ▶ O arquivo de implementação do módulo “ponto.c” deve sempre incluir o arquivo de interface do módulo (nesse caso ponto.h), isso é necessário por duas razões:
 - Podem existir definições na interface que são necessárias na implementação. Neste caso, precisamos da definição do tipo Ponto;
 - Garantirmos que as funções implementadas correspondam às funções da interface. Como o protótipo das funções exportadas é incluído, o compilador verifica por exemplo, se os parâmetros das funções implementadas equivalem aos parâmetros dos protótipos.
- ▶ Além da própria interface (biblioteca), é necessário incluir as interfaces das funções que usamos da biblioteca padrão.

Estrutura de Dados 1

Tipo Abstrato de Dado

► Arquivo ponto.c:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include "Ponto.h" //inclui os protótipos
5
6  //definição de tipos de dados
7  struct ponto{
8      float x;
9      float y;
10 };
11
12 //Aloca e retorna um ponto com coordenadas "x" e "y"
13 Ponto *pto_cria(float x, float y){
14     Ponto *p = (Ponto*) malloc(sizeof(Ponto));
15     if(p!=NULL){
16         p->x = x;
17         p->y = y;
18     }
19     return p;
20 }
```

Estrutura de Dados 1

Tipo Abstrato de Dado

```
22 //libera a memória alocada para um ponto
```

```
23 void pto_libera(Ponto *p){  
24     free(p);  
25 }
```

```
26  
27 //Recupera, por referência, o valor de um ponto
```

```
28 void pto_acessa(Ponto *p, float *x, float *y){  
29     *x = p->x;  
30     *y = p->y;  
31 }  
32
```

```
33 //Atribui a um ponto as coordenadas "x" e "y"
```

```
34 void pto_atribui(Ponto *p, float x, float y){  
35     p->x = x;  
36     p->y = y;  
37 }
```

```
38  
39 //Calcula a distância entre dois pontos
```

```
40 float pto_distancia(Ponto *p1, Ponto *p2){  
41     float dx = p1->x - p2->x;  
42     float dy = p1->y - p2->y;  
43     return sqrt(dx * dx + dy * dy);  
44 }
```

Arquivo ponto.c - continuação

Estrutura de Dados 1

Tipo Abstrato de Dado

► Arquivo principal, main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "Ponto.h"
4
5  int main(int argc, char *argv[])
6  {
7      float d;
8      Ponto *p, *q;
9      p = pto_cria(10, 21);
10     q = pto_cria(7, 25);
11     d = pto_distancia(p, q);
12     printf("Distancia entre os pontos: %f\n", d);
13     pto_libera(p);
14     pto_libera(q);
15     system("PAUSE");
16     return 0;
17 }
```

Façam dois testes no arquivo principal main():

- Tentem criar uma variável (que não seja ponteiro), comum do tipo Ponto;
- Tentem acessar um elemento dentro da estrutura "q"

Com a implementação deste TAD, temos a Atividade 2 completada. Entregue-a no Moodle.

Estrutura de Dados 1

Atividade 2

- Com a implementação deste TAD, temos a Atividade 2 completada. Entregue-a no Moodle como atividade 2.

Estrutura de Dados 1

Atividade 3

- ▶ Crie um TAD que efetue operações matemáticas tais como soma, subtração, multiplicação e divisão de valores, armazenando o resultado obtido. O TAD deve ter os dados encapsulados, e enquanto o programa funcionar ele deverá armazenar o resultado da operação anterior, **somente o resultado da última operação**, ou seja, apenas 1 campo.
- ▶ No programa principal, acrescente um menu onde o usuário insira novos dados para cálculo e escolha a operação a ser executada. O menu deverá funcionar continuamente e ter uma opção de acesso ao dado referente ao resultado da última operação realizada anteriormente. O menu deverá possuir também uma opção de encerramento do programa.
- ▶ Entregue no Moodle como atividade 3.