



Estruturas de Dados 1

09 – Tipo Abstrato de Dado - TAD

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, e que foi desenvolvido por outro programador, para o código que estamos desenvolvendo;
- Imagine se todos esses códigos aparecessem quando estamos desenvolvendo um programa. Um simples programa que gera um “hello world” teria centenas de linhas de código;
- O objetivo dessas bibliotecas ou **headers**, é o de criar e colocar em um arquivo, diversas funções específicas, que então são separadas e organizadas por assunto.



Estrutura de Dados 1



Criando Bibliotecas em C

- Para criação de uma biblioteca exemplo, utilizaremos o código de um exercício anterior:

```
#include <stdio.h>
#include <stdlib.h>

void funcao_iterativa(int n);
int funcao_rekursiva(int n);

int main(){
    int n;
    printf("Entre com um numero inteiro: ");
    scanf(" %d", &n);
    printf("Versao iterativa:\n");
    funcao_iterativa(n);
    printf("\n\n");
    printf("Versao recursiva:\n");
    //descarta valor devovido pois não é necessário
    funcao_rekursiva(n);
    printf("\n\n\n\n");
    system("pause");
    return 0;
}
```

Colocaremos
estas funções
em uma
biblioteca

```
void funcao_iterativa(int n){
    while(n >= 0){
        printf("\t %d \n", n);
        n--;
    }
}

int funcao_rekursiva(int n){
    printf("\t %d \n", n);
    if(n == 0){
        return 0;
    }
    return funcao_rekursiva(n - 1);
}
```

Estrutura de Dados 1

Criando Bibliotecas em C

- Antes, para nos certificar, rodamos o programa exemplo que será dividido em uma biblioteca:

```
C:\Users\angelot\Documents\Aulas\GRUEDA1\Aulas\Aula 09 - Tipo At
Entre com um numero inteiro: 5
Versao iterativa:
5
4
3
2
1
0

Versao recursiva:
5
4
3
2
1
0

Pressione qualquer tecla para continuar. . .
```

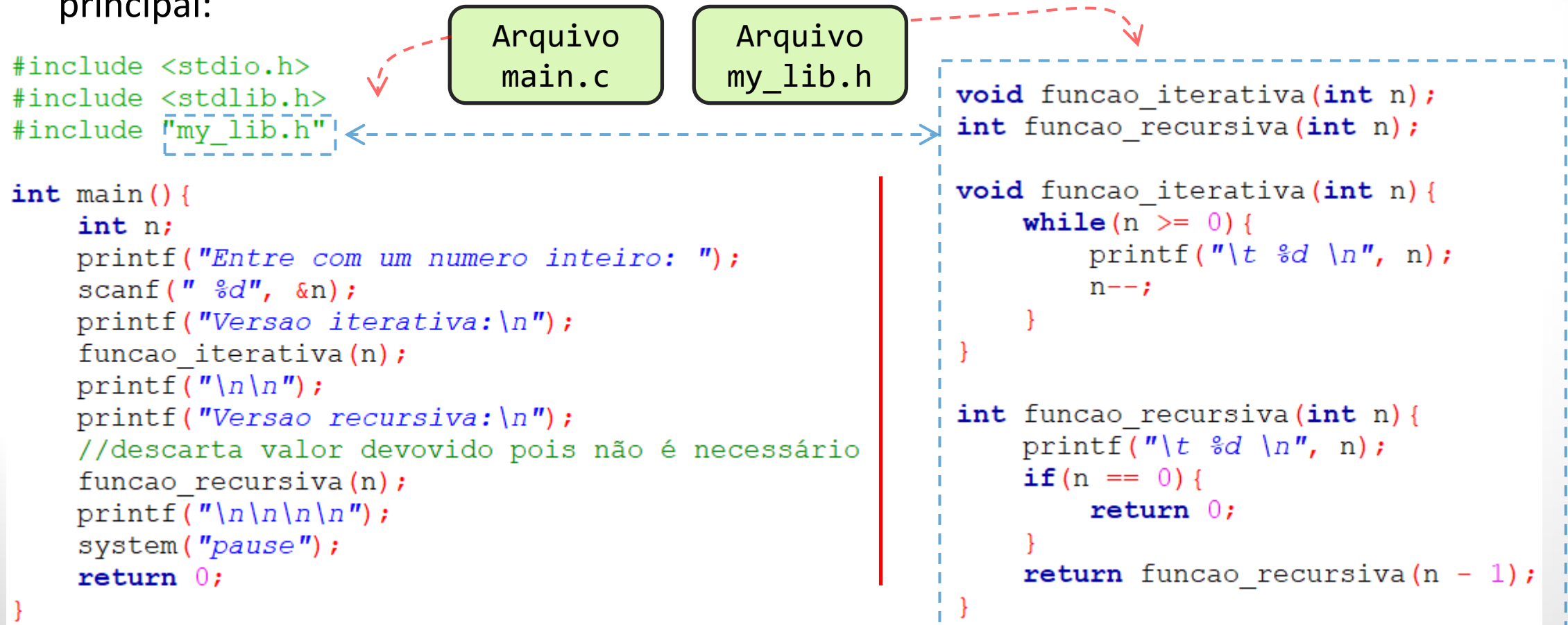


Estrutura de Dados 1



Criando Bibliotecas em C

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



Estrutura de Dados 1

Criando Bibliotecas em C

- Rodando o programa exemplo após a divisão de seu código em dois arquivos, `main.c` e `my_lib.h`:

```
C:\Users\angelot\Documents\Aulas\GRUEDA1\Aulas\Aula 09 - Tipo At
Entre com um numero inteiro: 5
Versao iterativa:
5
4
3
2
1
0

Versao recursiva:
5
4
3
2
1
0

Pressione qualquer tecla para continuar. . .
```



Estrutura de Dados 1

Criando Bibliotecas em C

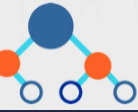
- Se você abrir um **Header**, os arquivos .h que costumamos usar, verá que eles contém diversos protótipos de funções, muitos comentários e outras coisas. Mas código C (funções), você verá muito pouco;
- Nos **Headers**, são declaradas todas as funções, bem como comentários a respeito delas, explicando o que são, para que servem, que parâmetros recebem e o que retornam;
- Os arquivos .h, são uma espécie de intermediário, entre o seu programa e os módulos de bibliotecas. Os arquivos “**Headers**” são o local onde é explicado o que cada função que foi definida nos módulos de biblioteca 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 o que aquele módulo de biblioteca tem a oferecer;
- Mas o mais importante: não precisamos saber de qual maneira as funções que pertencem àquela biblioteca específica foram implementadas. **Não precisamos conhecer o código interno de suas funções;**
- Ao contrário de ler um módulo com centenas de linhas de código para entender como as funções funcionam, 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, que você estivesse programando um jogo em C, com módulos de funções de áudio, imagem, fases, personagens, jogabilidade, etc. Seu módulo principal (`main.c`) ficaria impossível de se trabalhar, tamanho o número de protótipos e funções específicas para as várias funcionalidades diferentes. É aí que entram os arquivos **Headers** ou simplesmente `.h`;
- Colocamos neles, todos os protótipos, e ainda mais: colocaremos descrições simples diretas de como os recursos que compõem esta biblioteca funcionam, e essa é a parte importante. Porque é neste arquivo que o utilizador da biblioteca terá acesso às informações de como utilizar as funções que estão disponíveis na biblioteca.



Estrutura de Dados 1

Dividindo o programa em módulos

- Utilizando o mesmo exemplo anterior, onde criamos a biblioteca `my_lib.h` com os protótipos e funções, e em seguida fizemos a referenciamos esta biblioteca no programa principal, faremos mais uma mudança:
- Criaremos mais um arquivo de código com extensão `.c` (`my_lib.c`). Este arquivo deve ter o mesmo nome do arquivo `.h` (`my_lib.h`). Ele conterá as funções propriamente ditas;
- No arquivo `.h` deixaremos **somente os protótipos das funções** e colocaremos **comentários de como estes recursos funcionam**, o que recebem como parâmetro, o que retornam, etc. Assim, o arquivo `.h` funcionará como um manual de utilização para um programador, que por ventura, venha a incluir esta biblioteca em seu programa.



Estrutura de Dados 1

Criando projetos – CodeBlocks

- Mas para que tudo isso funcione corretamente, é necessária uma mudança na utilização da **IDE CodeBlocks**.
- Agora é necessário a criação de um **projeto** no ambiente de programação.
- Como são módulos (arquivos) separados, são compilados separadamente, portanto precisamos que sejam unidos ou ligados (*linked*) durante a compilação, para a geração do arquivo executável final. Inicializando um projeto dentro da IDE conseguimos essa ligação. Todos os arquivos que fazem parte do projeto serão então 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 efetuar essa ligação com programas específicos. Chamávamos estas operações de *linkedição*, ou *lincagem*.
- Podemos, usando nosso exemplo, compilar via terminal (ou console) os dois arquivos codificados separadamente, o `main.c` e o `my_lib.c`, em um arquivo executável em uma única linha de comando:

```
gcc -o prog.exe main.c my_lib.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 my_lib.c  
gcc -o prog.exe main.o my_lib.o
```

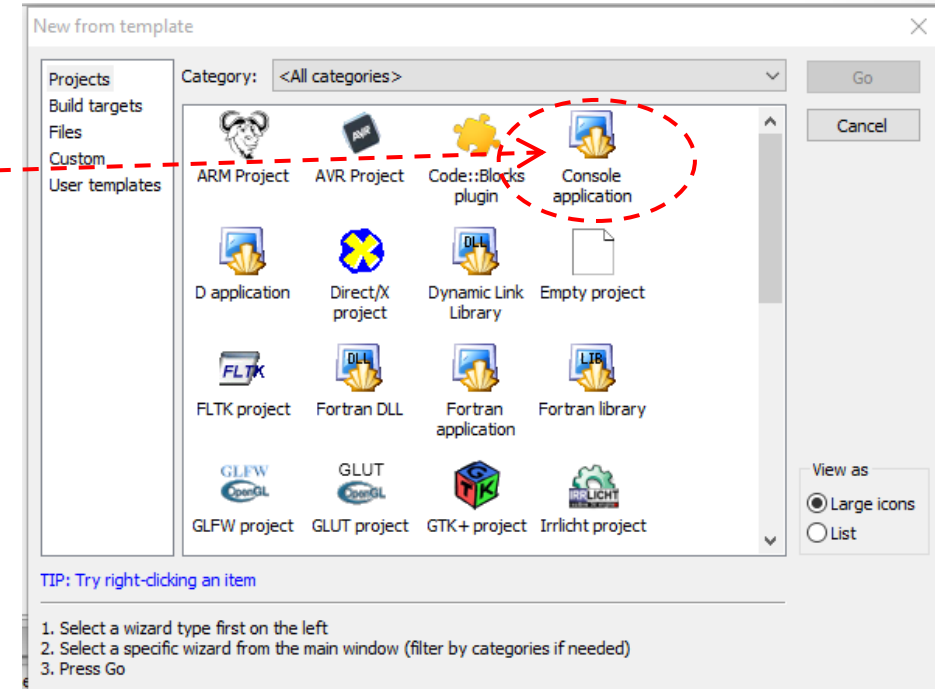
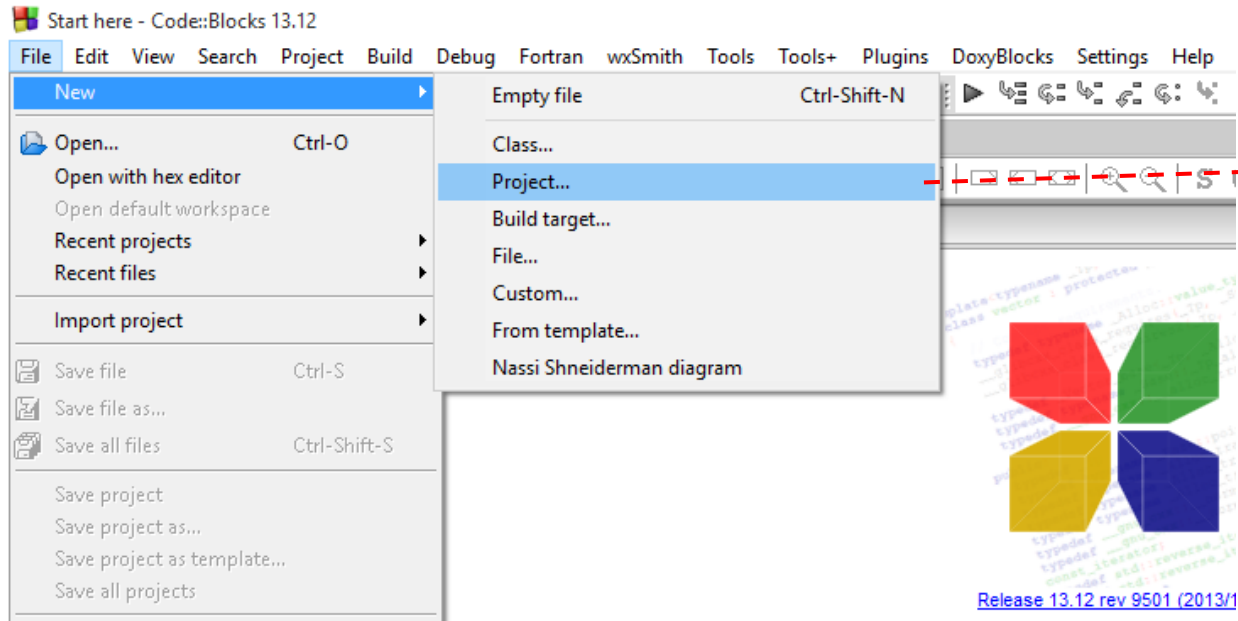
-o: especifica o nome do arquivo de saída
-c: compila o código fonte em um arquivo objeto com extensão .o



Estrutura de Dados 1

Criando projetos – CodeBlocks

- Compilando agora, utilizando a IDE CodeBlocks:



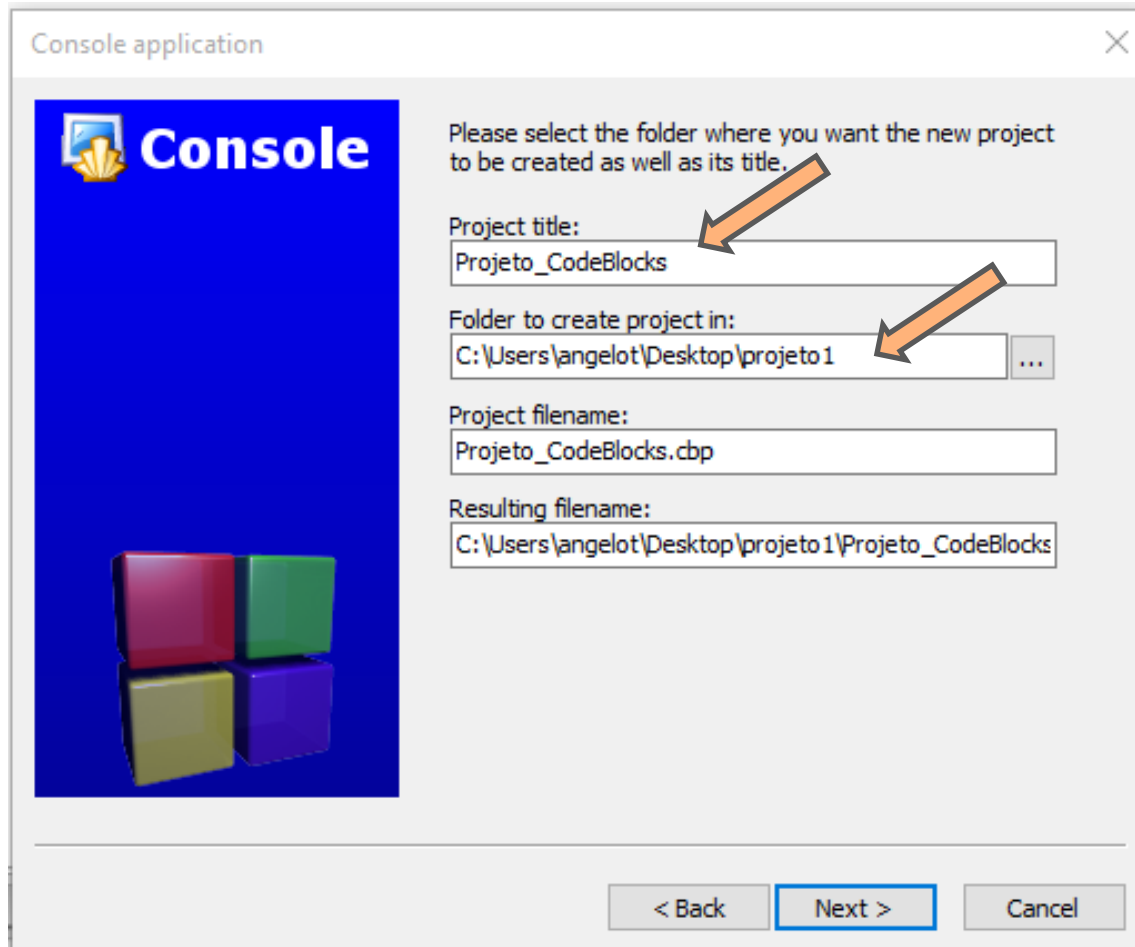
- Vá em: File/New/Project...
- Na nova janela, escolha: *Console application*.



Estrutura de Dados 1

Criando projetos – CodeBlocks

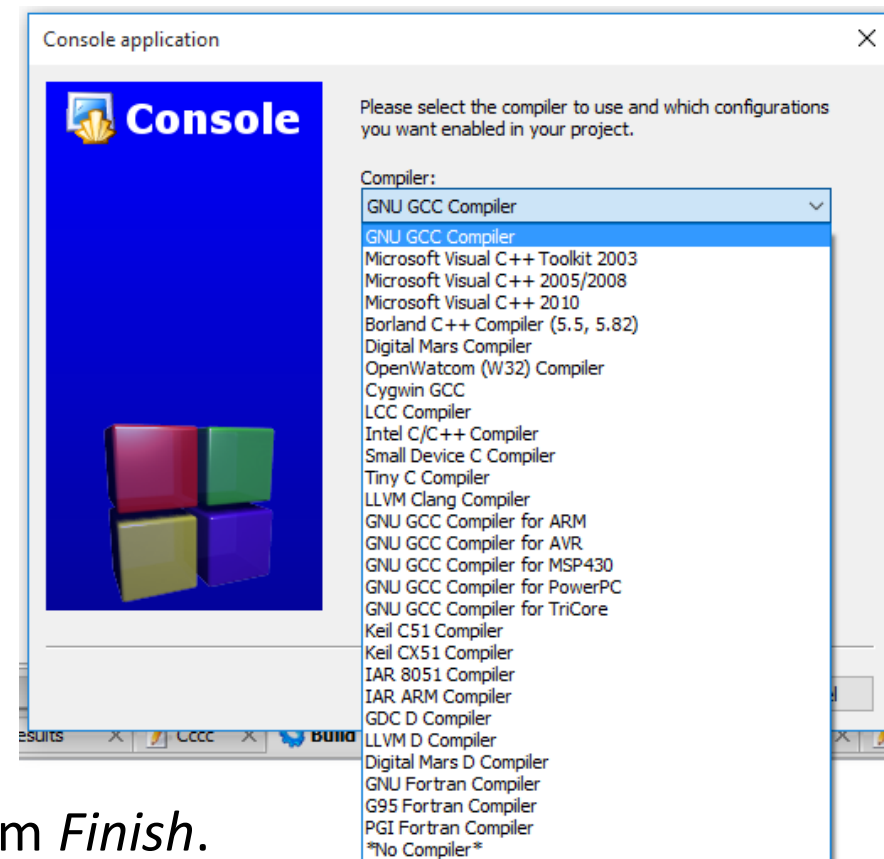
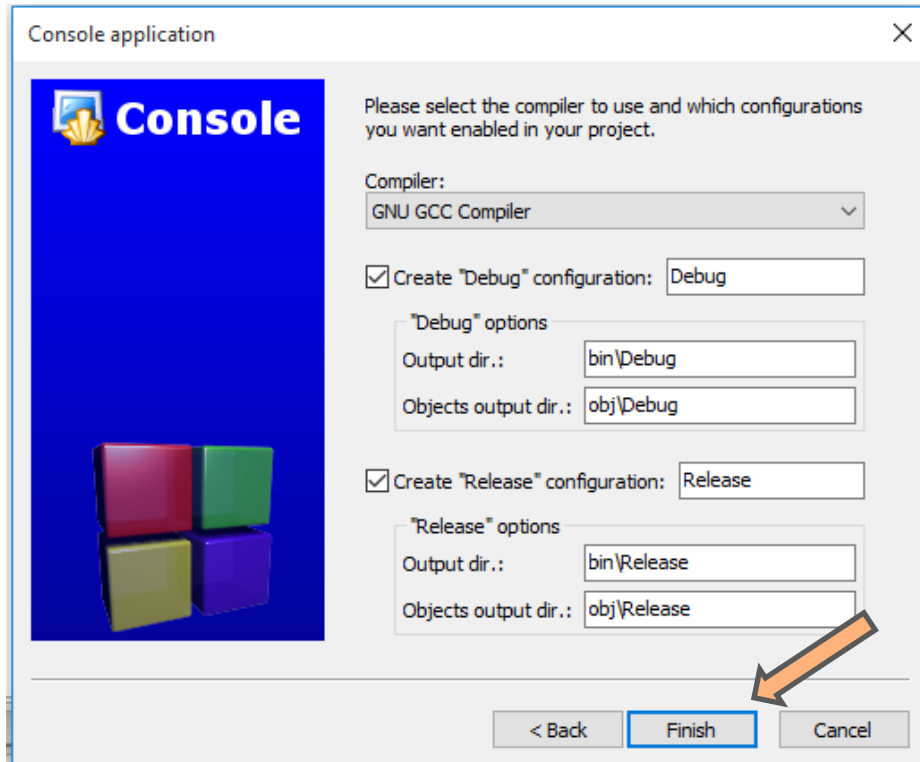
- Dê um nome ao seu novo projeto e escolha a pasta onde será salvo:



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:



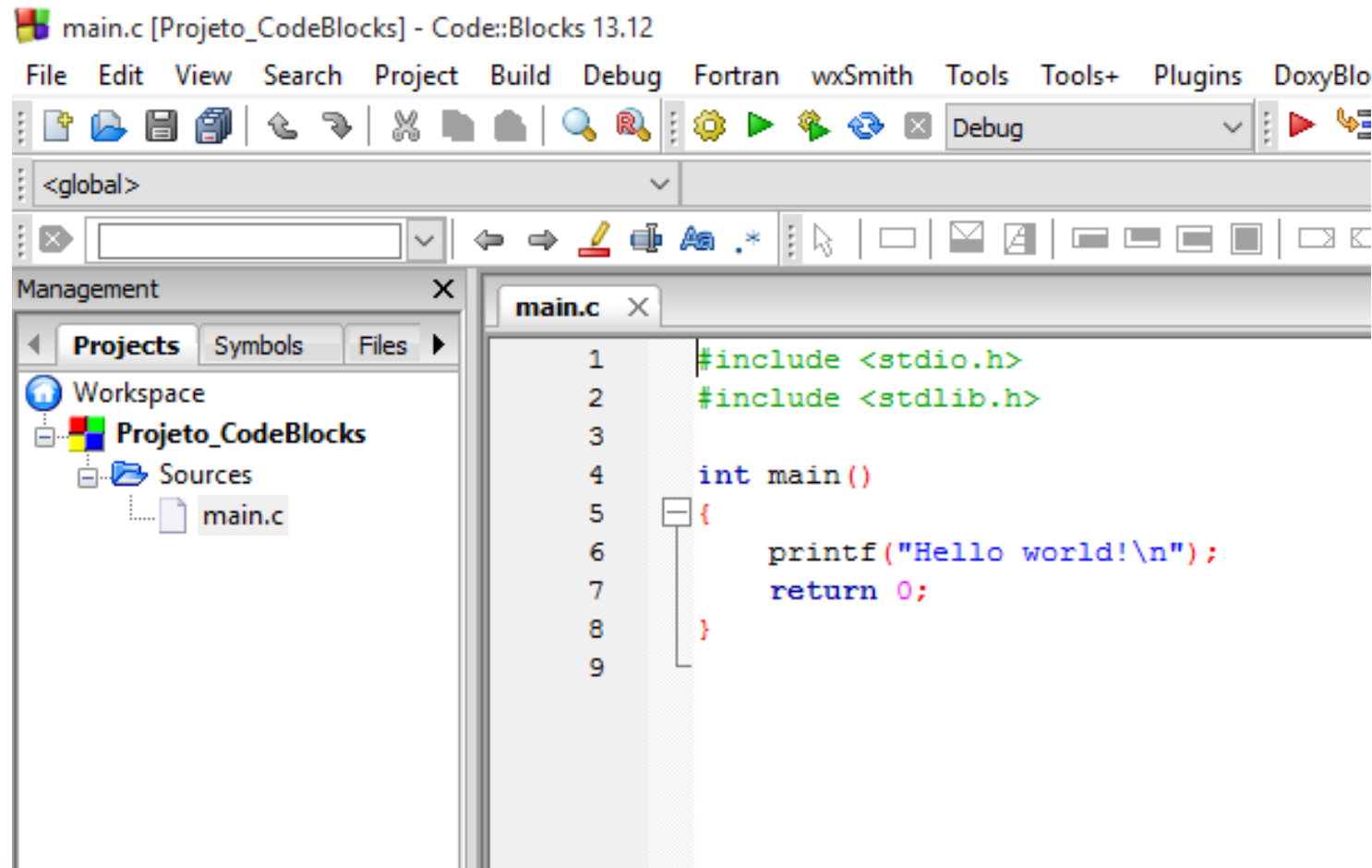
- Não sendo necessária nenhuma alteração, clique em *Finish*.



Estrutura de Dados 1

Criando projetos – CodeBlocks

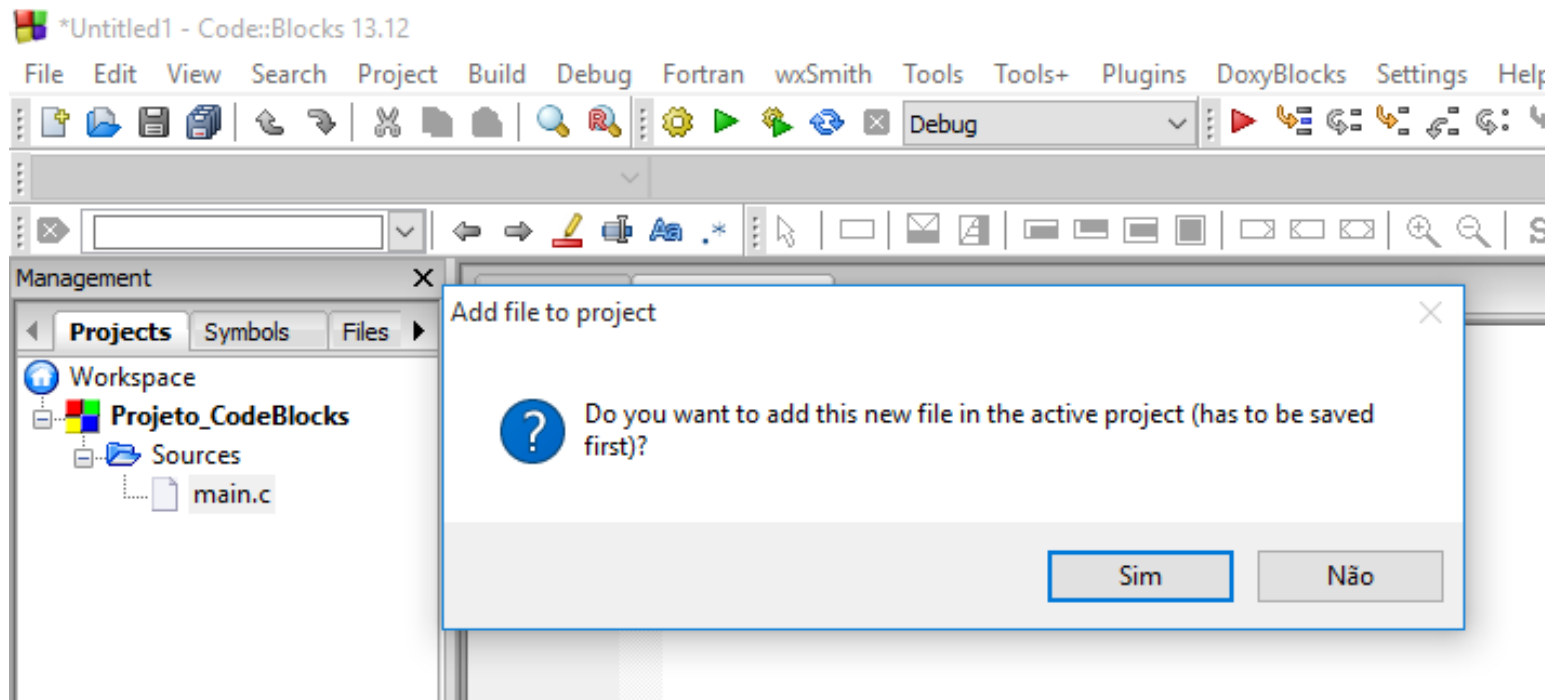
- Com o projeto criado, teremos esse resultado o *template* padrão do IDE:



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* como era feito antes. Mas agora, CodeBlocks irá perguntar se você quer adicioná-lo ao projeto atual, responda que sim:



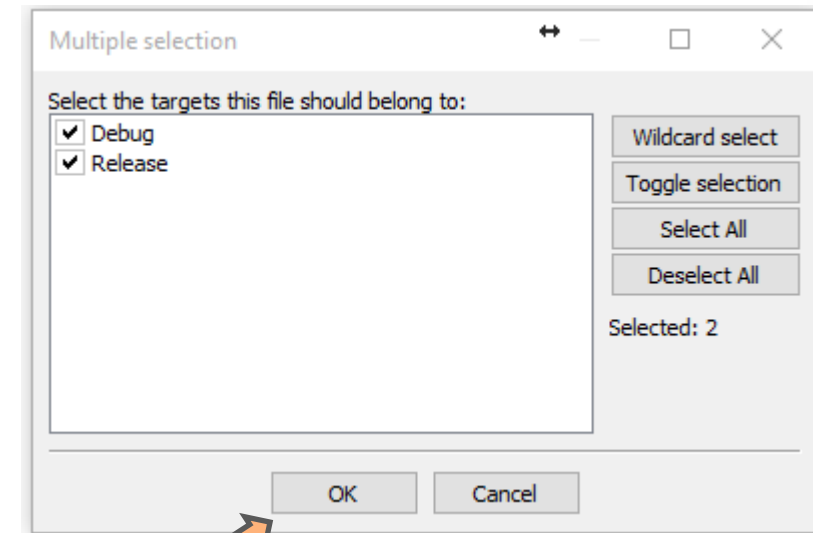
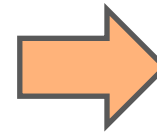
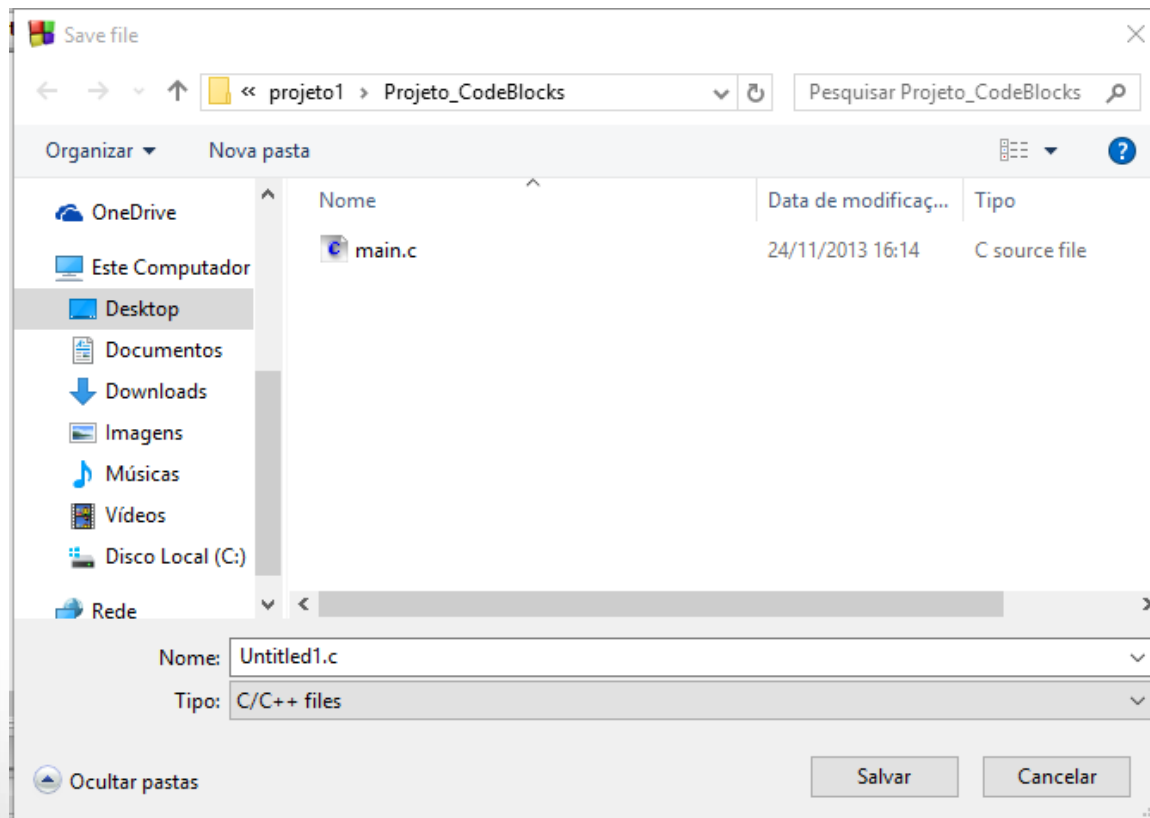
- Criaremos 2 arquivos para nosso projeto: “my_lib.h” e “my_lib.c”.



Estrutura de Dados 1

Criando projetos – CodeBlocks

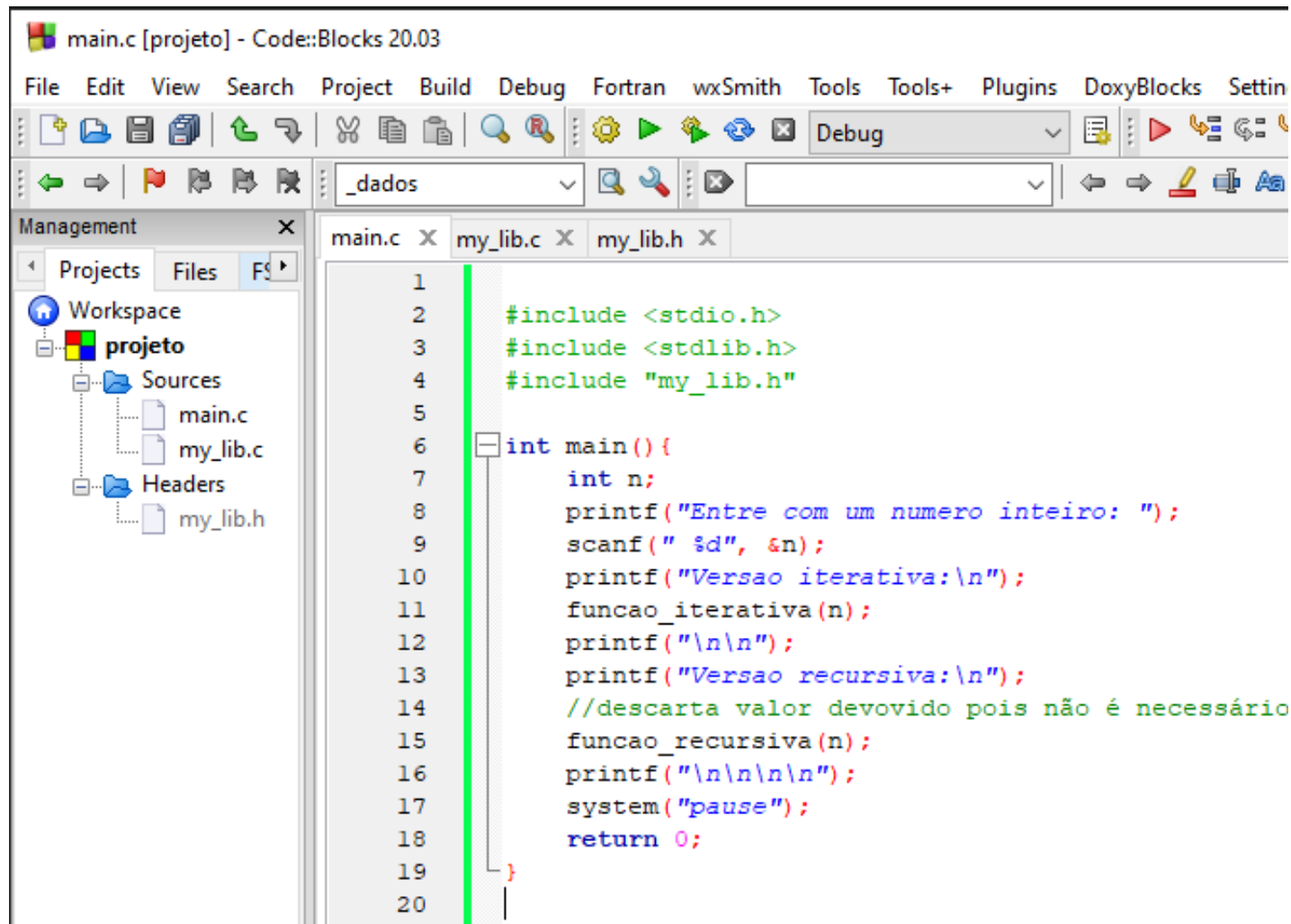
- Uma janela padrão do sistema será aberta, já direcionada para a pasta onde você criou o seu projeto, para que então o arquivo seja salvo. Nomeie-o “my_lib.h” e salve-o. Uma caixa de diálogo aparecerá, clique em ok. Em seguida repita toda a operação de criação para um segundo arquivo, e salve-o com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



Estrutura de Dados 1

Criando projetos – CodeBlocks

- Agora com os arquivos abertos na IDE, é só digitar os códigos no programa principal, main.c:



```
main.c [projeto] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settin
Debug
_dados
Management
Projects Files
Workspace
projeto
Sources
main.c
my_lib.c
Headers
my_lib.h
1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include "my_lib.h"
5
6 int main(){
7     int n;
8     printf("Entre com um numero inteiro: ");
9     scanf(" %d", &n);
10    printf("Versao iterativa:\n");
11    funcao_iterativa(n);
12    printf("\n\n");
13    printf("Versao recursiva:\n");
14    //descarta valor devovido pois não é necessário
15    funcao_recursiva(n);
16    printf("\n\n\n\n");
17    system("pause");
18    return 0;
19 }
20
```



Estrutura de Dados 1

Criando projetos – CodeBlocks

- O módulo que contém as funções “my_lib.c”, também deve ter as bibliotecas referenciadas:



my_lib.h [projeto] - Code::Blocks 20.03

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Debug

Management

Workspace

projeto

Sources

main.c

my_lib.c

Headers

my_lib.h

```
1
2
3  /* *****
4  * imprime o número passado, *
5  * decrementando-o até zero *
6  * de forma iterativa      *
7  * ***** */
8  void funcao_iterativa(int n);
9
10 /* *****
11 * imprime o número passado, *
12 * decrementando-o até zero *
13 * de forma recursiva      *
14 * ***** */
15 int funcao_recursiva(int n);
```

*my_lib.c [projeto] - Code::Blocks 20.03

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins Debug

Management

Workspace

projeto

Sources

main.c

my_lib.c

Headers

my_lib.h

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "my_lib.h"
4
5  void funcao_iterativa(int n){
6      while(n >= 0){
7          printf("\t %d \n", n);
8          n--;
9      }
10 }
11
12 int funcao_recursiva(int n){
13     printf("\t %d \n", n);
14     if(n == 0){
15         return 0;
16     }
17     return funcao_recursiva(n - 1);
18 }
```

Estrutura de Dados 1

Criando projetos – CodeBlocks

- Agora é só compilar, e confirmar o resultado:

```
"C:\Users\angelot\Documents\Aulas\GRUEDA1\Aulas\Aula 09 - Tipo Abstrado de Dado - TA"
Entre com um numero inteiro: 5
Versao iterativa:
    5
    4
    3
    2
    1
    0

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 - arquivo .h) como interface, e um módulo que conterà 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`:
 $-n \dots -2, -1, 0, 1, 2, \dots n$
- Definição de Estrutura de dados:
 - Conjunto de variáveis que possuem um relacionamento lógico entre seus tipos de dados, exemplo:

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

No contexto desta estrutura, as variáveis internas que a compõem, passam a ter um significado lógico entre si.



Estrutura de Dados 1

Tipo Abstrato de Dado

- Tipos abstratos de dados incluem (além da estrutura lógica para armazenamento dos seus dados), as operações (funções) específicas e exclusivas para manipulação de seus próprios dados.

Essas funções são a única forma de acesso aos dados que o TAD armazena!

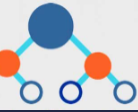
- Diferente da utilização simples da `struct`, onde podemos acessar seus campos livremente, não temos acesso direto ao que está dentro da estrutura que pertence ao TAD. O acesso aos seus dados só é permitido por meio das funções que foram definidas para trabalhar com este tipo abstrato:
 - Criação da estrutura – alocação de memória;
 - 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 internos (e este será o único meio), e dessa forma, estes dados ficam ocultos do programador/usuário que utilizará este TAD.



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 os dados internos do TAD por meio de suas funções, **nunca diretamente**.
- Reuso;
 - O TAD pode ser utilizado por diferentes programas.
- O TAD pode ser 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, criaremos um TAD que armazenará um ponto do plano cartesiano definido por suas coordenadas “x” e “y”

```
//definição de tipos de dados
struct pontoCartesiano{
    float x;
    float y;
};
```

1º passo para definir no arquivo “.h”:

- Protótipos das funções;
- Tipos de ponteiros;
- Dados globalmente acessíveis.



Estrutura de Dados 1

Tipo Abstrato de Dado

- Por convenção, em Linguagem C sempre usamos dois arquivos para implementar um TAD, que sempre são criados em módulos.
 - Arquivo “.h ” – Contém os protótipos das funções, **o tipo de ponteiro que apontará para o dado no módulo “.c ”**, dados que são globalmente acessíveis (por exemplo as constantes) e instruções de utilização;
 - Arquivo “.c ” – Contém a declaração de criação **do tipo de dado** (struct) que armazenará dados, que só será acessível através de seu endereço em memória, dessa forma se tornando um tipo de dado **OPACO**, e a implementação das funções que controlarão o acesso a estes dados opacos. Esta será a única forma de acesso aos dados, atingindo assim o **ENCAPSULAMENTO** dos dados (Dado opaco + funções que o controlam).
- 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 .



Estrutura de Dados 1

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 `PTcart`, e o conjunto de funções que operam sobre este tipo;
- Neste exemplo, vamos considerar as seguintes funções:
 - `criaPTcart()` – cria um ponto com coordenadas `x` e `y`;
 - `liberaPTcart()` – libera a memória alocada por um ponto;
 - `acessaPTcart()` – devolve as coordenadas de um ponto;
 - `atribuiPTcart()` – atribui novos valores às coordenadas de um ponto;
 - `distanciaPTcart()` – calcula a distância entre dois pontos.



Estrutura de Dados 1

Tipo Abstrato de Dado

- A interface (arquivo “PTcart.h”) é constituída pelo código:

```
//Arquivo PTcart.h
//Atribui novo nome para struct pontoCartesiano: PTcart
typedef struct pontoCartesiano PTcart;

//Cria um novo PTcart - somente ponteiro!!
PTcart *criaPTcart(float x, float y);

//Libera um PTcart
void liberaPTcart(PTcart *p);

//Acessa valores "x" e "y" de um PTcart
void acessaPTcart(PTcart *p, float *x, float *y);

//Atribui os valores "x" e "y" a um PTcart
void atribuiPTcart(PTcart *p, float x, float y);

//Calcula a distância entre dois Pontos no plano cartesiano
float distanciaPTcart(PTcart *p1, PTcart *p2);
```

Note que a composição da estrutura PTcart (struct pontoCartesiano) não é exportada pelo módulo. Dessa forma os demais módulos que usarem este TAD não poderão acessar diretamente os campos dessa estrutura, uma vez que só possuem o seu endereço na memória, e não têm acesso ao modelo da struct. Os clientes desse TAD só terão acesso às informações que possam ser obtidas através das funções exportadas pelo arquivo PTcart.h



Estrutura de Dados 1

Tipo Abstrato de Dado

- O arquivo de implementação do módulo “PTcart.c” deve sempre incluir o arquivo de interface do módulo (que nesse caso é o PTcart.h), isso é necessário por duas razões:
 - Podem existir definições na interface que são necessárias na implementação. Neste caso, precisamos da definição do tipo PTcart (o typedef);
 - 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, ou “.h”), é necessário incluir as interfaces das funções que usaremos das bibliotecas padrão.



Estrutura de Dados 1

Tipo Abstrato de Dado

- Arquivo PTcart.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "PTcart.h" //inclui os protótipos

//definição de tipos de dados - este modelo não é exportado
struct pontoCartesiano{
    float x;
    float y;
};

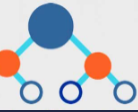
//Aloca e retorna um PTcart com coordenadas "x" e "y"
PTcart *criaPTcart(float x, float y)
{
    PTcart *p = (PTcart*) malloc(sizeof(PTcart));
    if(p!=NULL){
        p->x = x;
        p->y = y;
    }
    return p;
}

//libera a memória alocada para um PTcart
void liberaPTcart(PTcart *p){
    free(p);
}

//Recupera, por referência, o valor de um PTcart
void acessaPTcart(PTcart *p, float *x, float *y){
    *x = p->x;
    *y = p->y;
}

//Atribui a um PTcart as coordenadas "x" e "y"
void atribuiPTcart(PTcart *p, float x, float y){
    p->x = x;
    p->y = y;
}

//Calcula a distância entre dois PTcarts
float distanciaPTcart(PTcart *p1, PTcart *p2){
    float dx = p1->x - p2->x;
    float dy = p1->y - p2->y;
    return sqrt(dx * dx + dy * dy);
}
```



Estrutura de Dados 1

Tipo Abstrato de Dado

- Arquivo principal, main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "PTcart.h"

int main() {
    float d = 0, retornaX, retornaY;
    PTcart *p = NULL, *q = NULL;
    //PTcart r; <-----
    p = criaPTcart(10, 21);
    q = criaPTcart(7, 25);
    d = distanciaPTcart(p, q);
    //p.y = 3;
    //q.x = 2.34; <-----
    printf("Distancia entre os pontos cartesianos: %f\n", d);
    atribuiPTcart(q, 15, -2);
    d = distanciaPTcart(p, q);
    printf("Nova distancia entre os pontos cartesianos: %f\n", d);
    acessaPTcart(p, &retornaX, &retornaY);
    printf("Valores armazenados em p: X = %.2f e Y = %.2f\n", retornaX, retornaY);
    acessaPTcart(q, &retornaX, &retornaY);
    printf("Valores armazenados em q: X = %.2f e Y = %.2f\n", retornaX, retornaY);
    liberaPTcart(p);
    liberaPTcart(q);
    system("PAUSE");
    return 0;
}
```

Façam dois testes no arquivo principal main():

- Tentem criar uma variável comum (que não seja um ponteiro), do tipo PTcart;
- Tentem acessar um elemento dentro das estruturas “p” e “q”.

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



Estrutura de Dados 1

Tipo Abstrato de Dado



- Execução do programa:

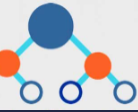
```
"C:\Users\angelot\Documents\Aulas\GRUEDA1\Aulas\Aula 09 - Tipo Abstrado de Dado - TAD\material de apoio\Tad
Distancia entre os pontos cartesianos: 5.000000
Nova distancia entre os pontos cartesianos: 23.537205
Valores armazenados em p: X = 10.00 e Y = 21.00
Valores armazenados em q: X = 15.00 e Y = -2.00
Pressione qualquer tecla para continuar. . .

Process returned 0 (0x0)   execution time : 20.701 s
Press any key to continue.
```

Estrutura de Dados 1

Atividade 2

- Com o exemplo do ponto cartesiano montado, a Atividade 2 estará completada. Entregue-a no Moodle como atividade 2.



Estrutura de Dados 1

Atividade 3

- Crie um TAD que efetue operações matemáticas com as 4 operações básicas, armazenando sempre o último resultado obtido. O TAD deve ter os dados encapsulados, e, enquanto o programa estiver rodando, deverá armazenar o resultado da operação anterior, **somente o resultado da última operação**, ou seja, apenas 1 resultado, consequentemente utilizará apenas 1 campo para isso.
- No programa principal, acrescente um menu onde o usuário insira novos valores 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.

