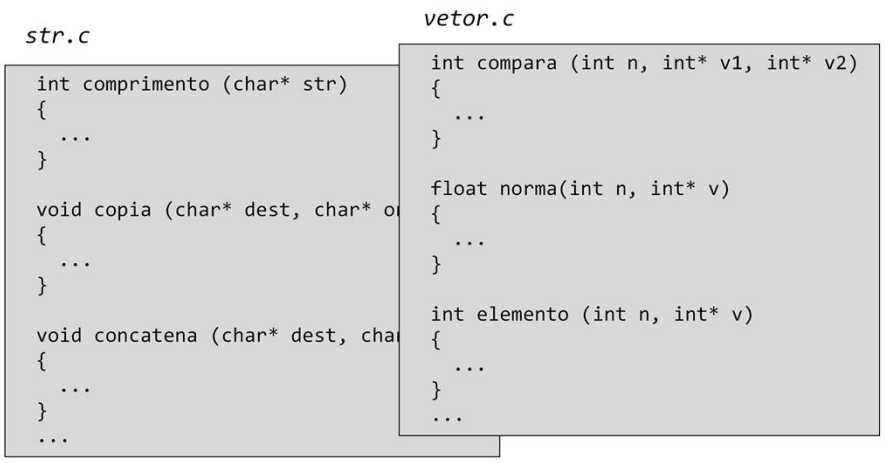
Programação 2

Tipos Abstratos

Rivera

Tipos Abstratos de Dados

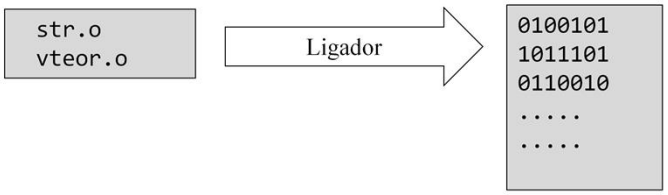
- Módulos
  - Um arquivo com funções relacionados (parte de um programa completo)



um módulo é uma parte de programa completo, onde são exportadas suas funções para um programa total. Esse é o caso de algumas bibliotecas, que ao serem importadas, adicionam seu módulo ao programa em questão, possuindo diversas funções já configuradas, facilitando a codificação

Tipos Abstratos de Dados

- Arquivo objeto
  - Resultado de compilar um módulo
  - Geralmente com extensão *.o ou *.obj
- Ligador
  - Junta todos os arquivos objetos e transforma em um único arquivo executável.



módulos podem ser compilados em arquivos objetos, de extensão .o ou .obj, e podem ser ligados posteriormente a outros programas objetos, criando um arquivo executável único

Módulos

- Exemplo

  Prog1.c

```
#include <stdio.h>
int comprimento (char* str);
void copia (char* dest, char* orig);
void concatena (char* dest, char* orig);

int main (void)
{
  char str[101], atr1[51], str2[51];
  printf("Digite uma seqüência de caracteres: ");
  scanf(" %50[^\n]", str1);
  printf("Digite outra seqüência de caracteres: ");
  scanf(" %50[^\n]", str2);
  copia(str, str1);
  concatena(str, str2);
  printf("Comprimento da concatenação: %d\n",comprimento(str));
  return 0;
}
```

as funções copia e concatena são de um módulo externo ao programa, porém como não há importação, as funções são declaradas, com seus tipos de dados, para que depois o arquivo prog1.o seja ligado ao str.o, e as funções possam funcionar adequadamente

Módulos

- Exemplo

  Prog1.exe

  1) Compilar fontes str.c e prog1.c separadamente
  2) Ligar os arquivos resultantes em um único arquivo

```
> gcc –c str.c –o str.o
> gcc –c prog1.c –o prog1.o
> gcc –o prog.exe str.o prog1.o
```

os arquivos podem ser compilados separadamente, em programas objetos, e após isso, ligados em um executável único.

Módulos

- Interfaces de um módulo de funções
  - Arquivo contendo apenas;
    - Os protótipos de funções oferecidas pelo módulo
    - Os tipos de dados exportados pelos módulos
  - Em geral possui:
    - Nome: o mesmo do módulo ao qual está associado
    - Extensão: *.h
  - Exemplo: str.h

```
int comprimento (char* str);
void copia (char* dest, char* orig);
void concatena (char* dest, char* orig);
```

a biblioteca str.h introduz várias funções de manipulação de caracteres e strings, e pode ser utilizada ao importar seu módulo `#include <str.h>`, compartilhando suas funções e os tipos de dados exportados pelo módulo

Módulos

- Exemplo

  Prog1.c

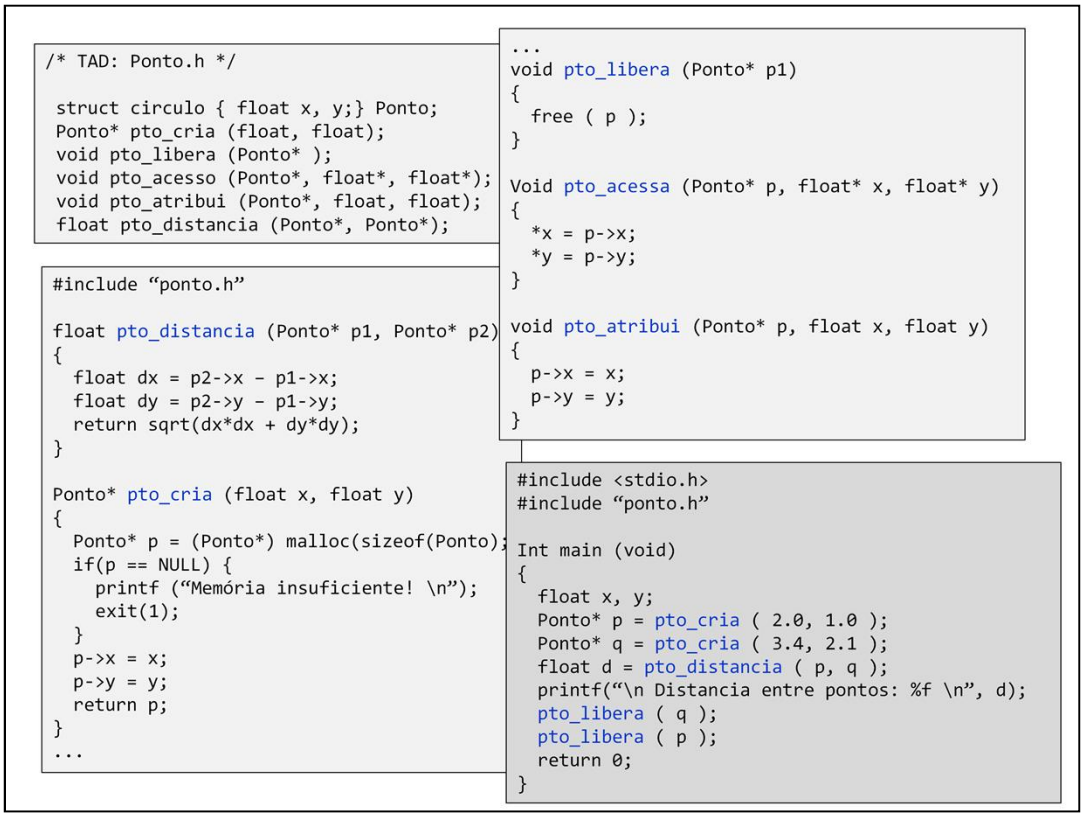
```
#include <stdio.h>   // protótipos das funções da bib padrão de C

#include “str.h”     // protótipos de módulos do usuário

int main (void)
{
  char str[101], atr1[51], str2[51];
  printf("Digite uma seqüência de caracteres: ");
  scanf(" %50[^\n]", str1);
  printf("Digite outra seqüência de caracteres: ");
  scanf(" %50[^\n]", str2);
  copia(str, str1);
  concatena(str, str2);
  printf("Comprimento da concatenação: %d\n",comprimento(str));
return 0;
}
```

o programa acima utiliza as funções já definidas dentro do módulo str.h, facilitando a codificação e implementação de funcionalidades, bastando apenas importar o módulo desejado.

Tipo Abstrato de Dados

- TAD
  - Um TAD define
    - Um novo tipo de dado
    - O conjunto de operações para manipular dados desse tipo
  - Um TAD facilita
    - A manipulação e a reutilização de código
    - Abstrato: "forma de implementação não precisa ser conhecida"
      - Para usar, conhecer
        - » Funcionalidade, mas não a sua implementação

Um tipo abstrato nada mais é do que um novo nome para um novo dado, visando funcionalidade e reutilização de código, dispensando a implementação.



```
/* TAD: Ponto.h */

 struct circulo { float x, y;} Ponto;
 Ponto* pto_cria (float, float);
 void pto_libera (Ponto* );
 void pto_acesso (Ponto*, float*, float*);
 void pto_atribui (Ponto*, float, float);
 float pto_distancia (Ponto*, Ponto*);
```

```
#include “ponto.h”

float pto_distancia (Ponto* p1, Ponto* p2)
{
  float dx = p2->x – p1->x;
  float dy = p2->y – p1->y;
  return sqrt(dx*dx + dy*dy);
}

Ponto* pto_cria (float x, float y)
{
  Ponto* p = (Ponto*) malloc(sizeof(Ponto));
  if(p == NULL) {
    printf (“Memória insuficiente! \n”);
    exit(1);
  }
  p->x = x;
  p->y = y;
  return p;
}
...
```

```
...
void pto_libera (Ponto* p1)
{
  free ( p );
}

Void pto_acessa (Ponto* p, float* x, float* y)
{
  *x = p->x;
  *y = p->y;
}

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

```
#include <stdio.h>
#include “ponto.h”

Int main (void)
{
  float x, y;
  Ponto* p = pto_cria ( 2.0, 1.0 );
  Ponto* q = pto_cria ( 3.4, 2.1 );
  float d = pto_distancia ( p, q );
  printf(“\n Distancia entre pontos: %f \n”, d);
  pto_libera ( q );
  pto_libera ( p );
  return 0;
}
```

a estrutura circulo (float x, float y) é introduzida, e recebe o nome de tipo Ponto. logo, todo o código é facilitado, uma vez que não precisa se definir a estrutura novamente, apenas fazer uso desse tipo abstrato, manipulando os dados dentro da estrutura, que inclusive possui funções já definidas pra isso, dentro do ponto.h

matriz.h

```
/* TAD: matriz m x n  como vetor */
typedef struct matriz
    {int lin; int col; float* v} Matriz;

Matriz* mat_cria (int m, int n);
void mat_libera (Matriz* mat);
float mat_acessa (Matriz* mat, int i, int j);
void mat_atribui (Matriz* mat, int i, int j, float v);
int mat_linhas (Matriz* mat);
int mat_colunas (Matriz* mat);
```

Implementar as funções.

matriz.h

```
/* TAD: matriz m por n */
typedef struct matriz
    {int lin; int col; float** vv} Matriz;

Matriz* mat_cria (int m, int n);
void mat_libera (Matriz* mat);
float mat_acessa (Matriz* mat, int i, int j);
void mat_atribui (Matriz* mat, int i, int j, float v);
int mat_linhas (Matriz* mat);
int mat_colunas (Matriz* mat);
```

a estrutura de matriz é definida como um tipo, e com isso, são implementadas funções que manipulam diretamente dados desse tipo, para realizar funções como liberar, atribuir, acessar, etc, facilitando a legibilidade do código, e a reutilização, uma vez que as funções e tipos já estão definidos, e só precisam ser chamados utilizando parâmetros necessários.