

数据结构绪论

斐多课堂  数据结构  第一讲
Phaedo Classes



3大模块



17道题目



数据结构绪论

模块1 / 数据结构的概念

模块2 / 在C语言中的数据类型

模块3 / 算法与算法分析

数据结构的概念

小节1 / 数据结构的基本概念

小节2 / 数据结构的三要素

数据结构的概念

小节1 / 数据结构的基本概念

小节2 / 数据结构的三要素

数据结构



数据结构

数据结构是相互之间存在一种或多种特定关系的**数据元素**的集合。

数据结构研究的问题

数据结构是一门研究**非数值计算**的程序设计问题中计算机的操作对象以及它们之间的关系和操作等的学科。

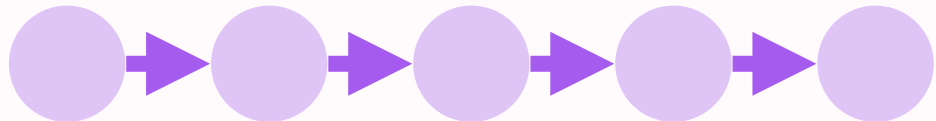
形式定义的数据结构

一个数据结构的形式定义，可描述为是一个二元组 $Data_Structures=(D,S)$ 。

其中： D 是数据元素的有限集， S 是 D 上关系的有限集。

设两个元素 x 和 y ，则可以用 $\langle x, y \rangle$ 表示由 x 指向 y ，用 (x, y) 表示由 x 和 y 有特定关系。

举例： $D = \{a_i | a_1, a_2, a_3, a_4, a_5\}$ ， $S_1 = \{ \langle a_1, a_2 \rangle, \langle a_2, a_3 \rangle, \langle a_3, a_4 \rangle, \langle a_4, a_5 \rangle \}$ ，

可以得到形如  的数据结构。

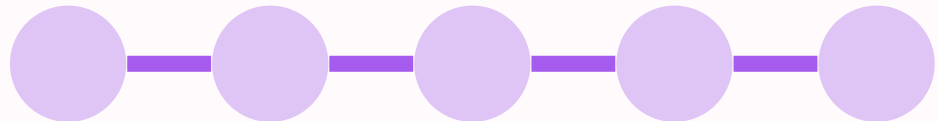
形式定义的数据结构

一个数据结构的形式定义，可描述为是一个二元组 $Data_Structures=(D,S)$ 。

其中： D 是数据元素的有限集， S 是 D 上关系的有限集。

设两个元素 x 和 y ，则可以用 $\langle x, y \rangle$ 表示由 x 指向 y ，用 (x, y) 表示由 x 和 y 有特定关系。

举例： $D = \{a_i | a_1, a_2, a_3, a_4, a_5\}$ ， $S_2 = \{(a_1, a_2), (a_2, a_3), (a_3, a_4), (a_4, a_5)\}$ ，

可以得到形如  的数据结构。



抽象数据类型

抽象数据类型是指一个数学模型以及定义在该模型上的一组操作，可用三元组 (D, S, P) 表示。

其中： D 是数据元素的有限集， S 是 D 上关系的有限集， P 是对 D 的基本操作集。

举例： $D = \{a_i | a_1, a_2, a_3, a_4, a_5\}$ ， $S_1 = \{ \langle a_1, a_2 \rangle, \langle a_2, a_3 \rangle, \langle a_3, a_4 \rangle, \langle a_4, a_5 \rangle \}$

$P =$ 删除第 i 个元素并返回其元素值 e ，

若 $i = 3$ 可以使数据结构 ，变为 。

抽象数据类型ADT

定义**ADT**：格式不唯一，可采用如下格式：



ADT <ADT名>

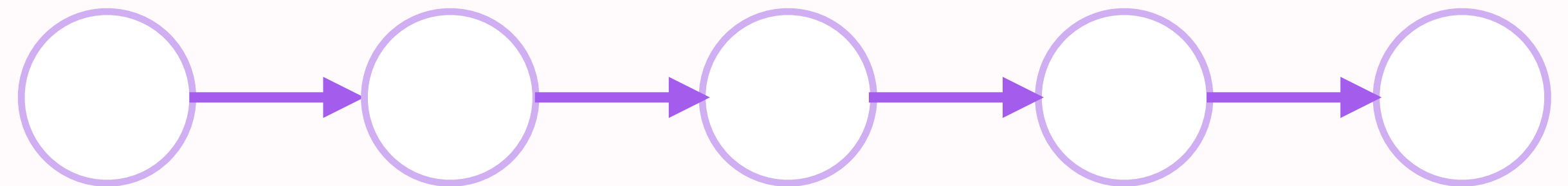
{

数据对象：<数据对象的定义>

结构关系：<结构关系的定义>

基本操作：<基本操作的定义>

}ADT <ADT名>



例题1-1 / 可以用（ ）定义一个完整的数据结构。

A.数据元素

B.数据对象

C.数据关系

D.抽象数据类型

解析1-1 / **D**

抽象数据类型（ADT）描述了数据的逻辑结构和抽象运算，通常用（数据对象，数据关系，基本类型操作）这样的三元组来表示，从而构成一个完整的数据结构定义。

例题1-2 / 在存储数据时，通常不仅要存储各数据元素的值，而且要存储（ ）。

A.数据的操作方法

B.数据元素的类型

C.数据元素之间的关系

D.数据的存取方法

解析1-1 / **C**

在存储数据时，不仅要存储数据元素的值，而且要存储数据元素之间的关系。

数据结构的概念

小节1 / 数据结构的基本概念

小节2 / 数据结构的三要素

数据结构的三要素

逻辑结构：是对数据元素之间的逻辑关系。

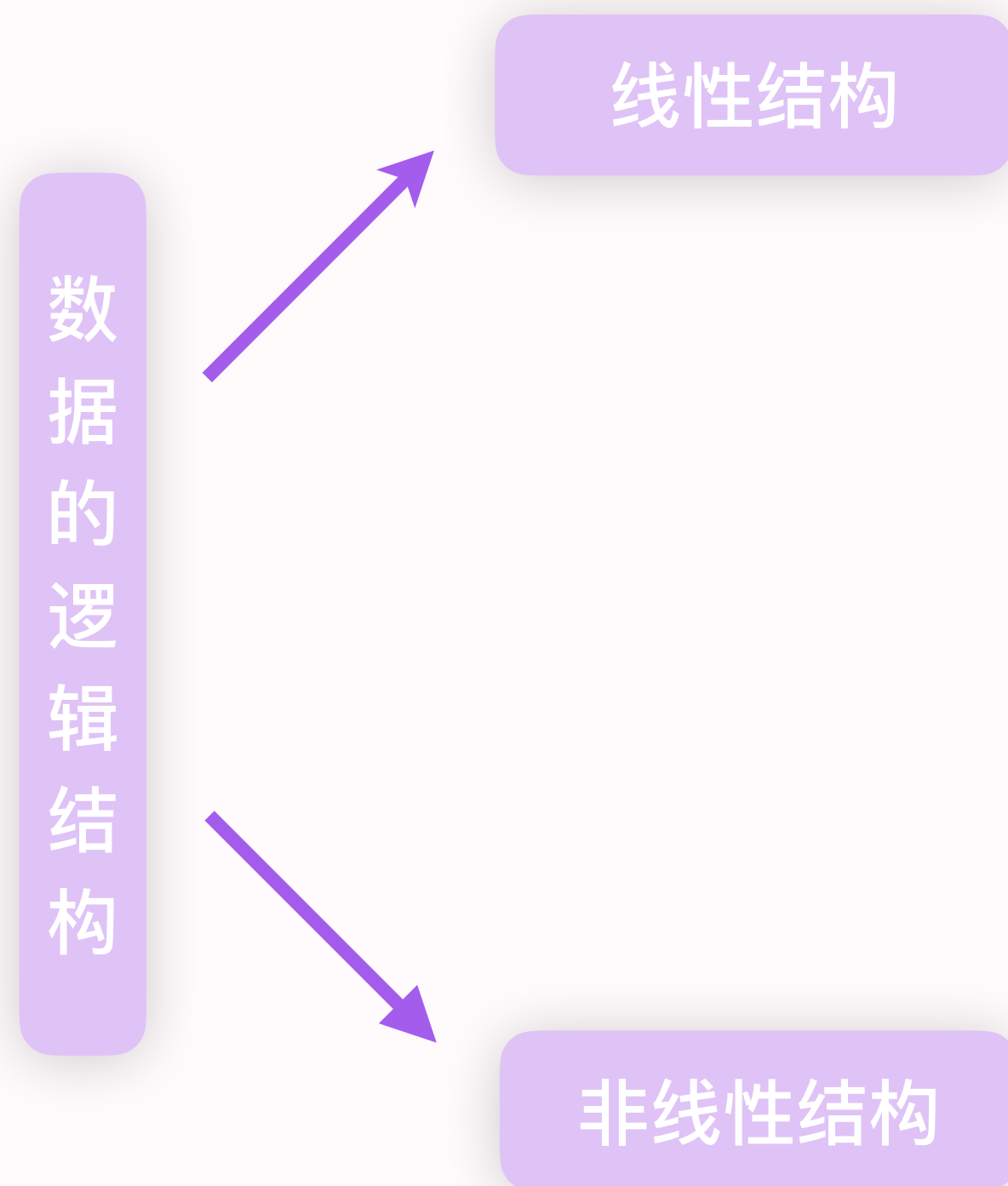
物理结构：是数据结构在计算机中的表示和实现，故又称“**存储结构**”。

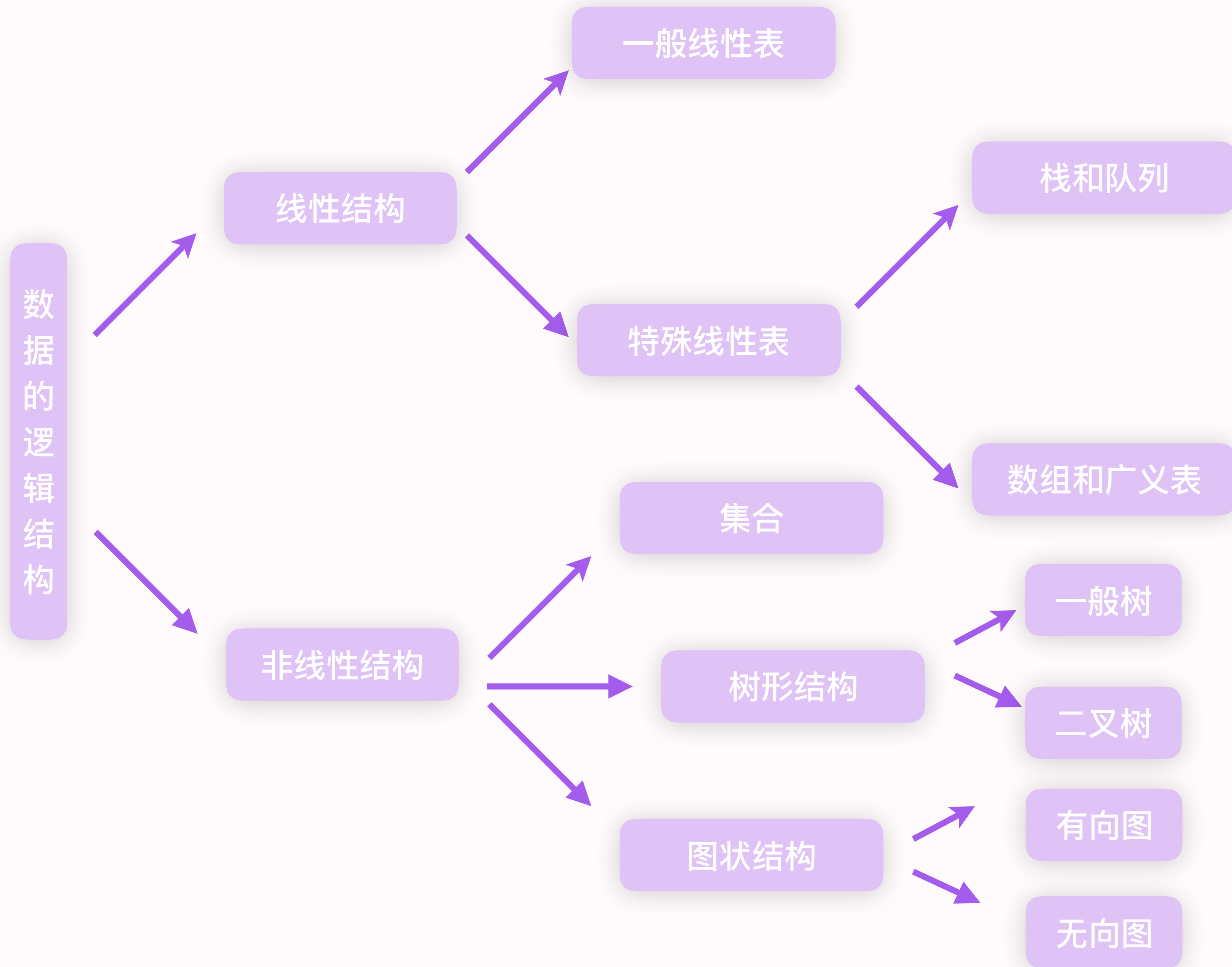
数据的运算：施加在数据上的运算包括运算的定义和实现。运算的定义是针对逻辑结构的，运算的实现是针对存储结构的。

数据的逻辑结构分类图

数据的逻辑结构



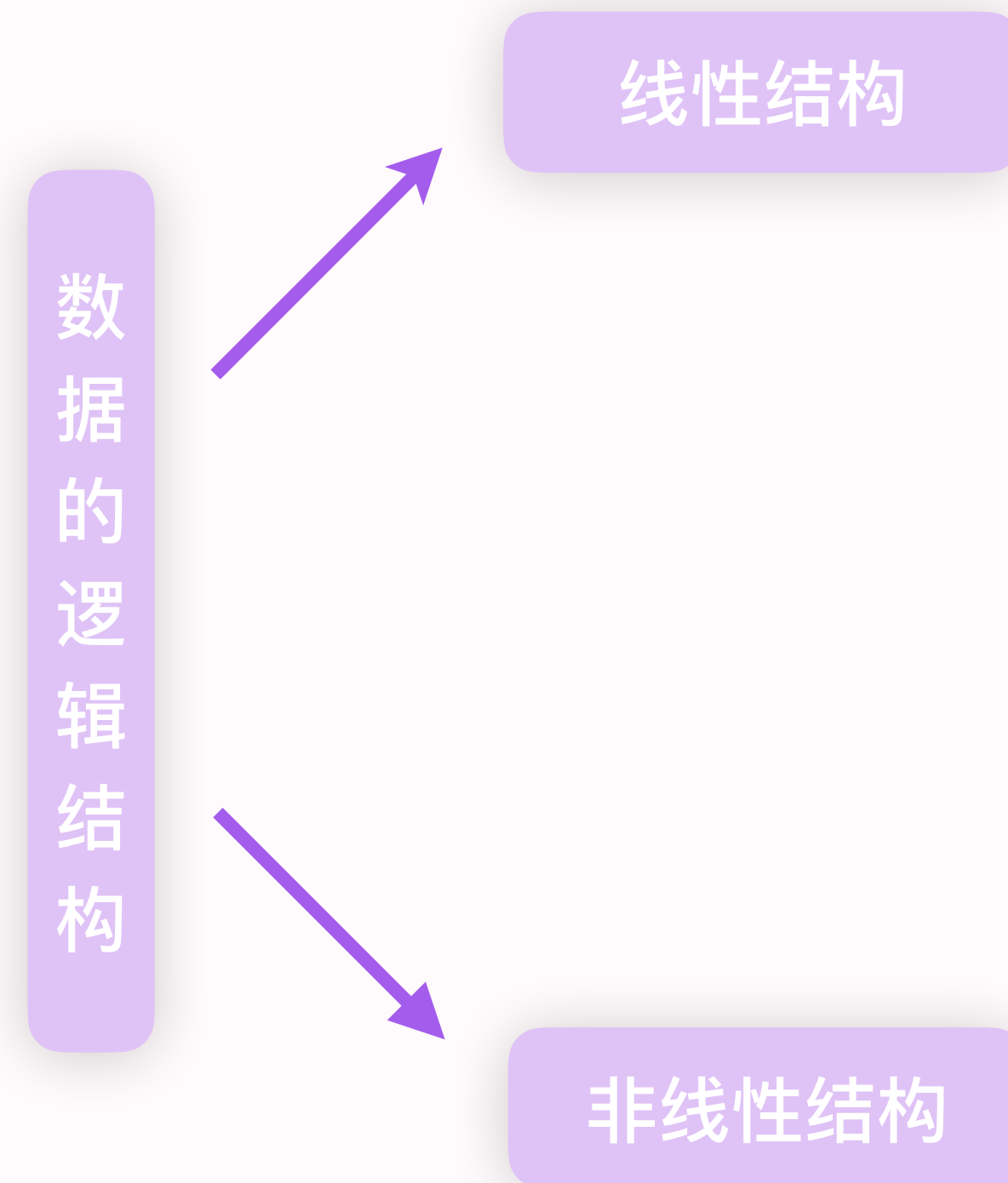




例题1-3 / 从逻辑上可以把数据结构分为（ ）两大类

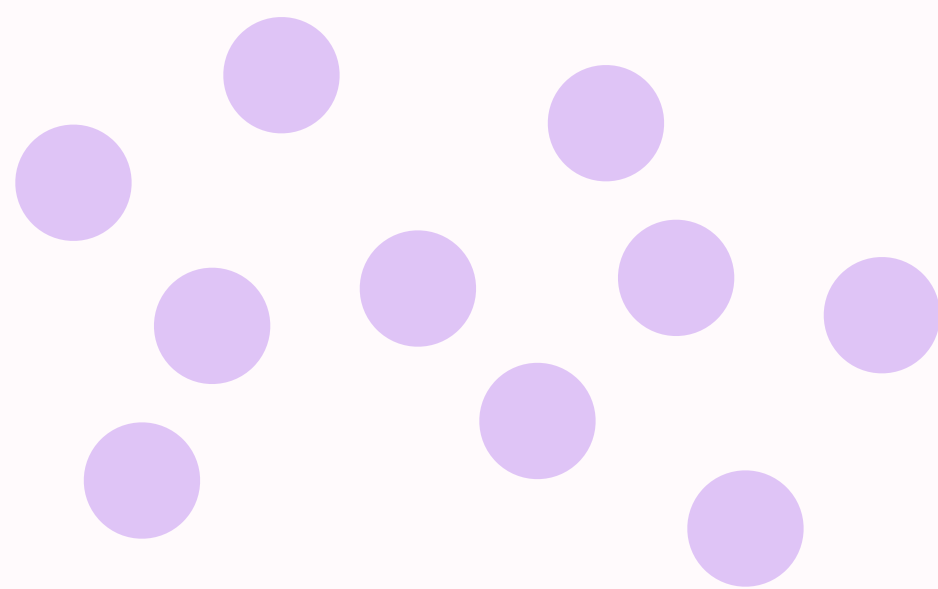
- A. 动态结构、静态结构
- B. 顺序结构、链式结构
- C. 线性结构、非线性结构
- D. 初等结构、构造型结构

解析1-3 / C



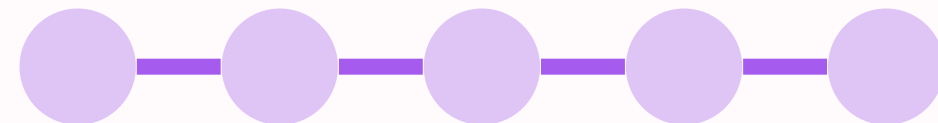
逻辑结构的四大分类

根据数据元素之间关系的不同特性，分为集合、线性结构、树状结构、图状或网状结构。



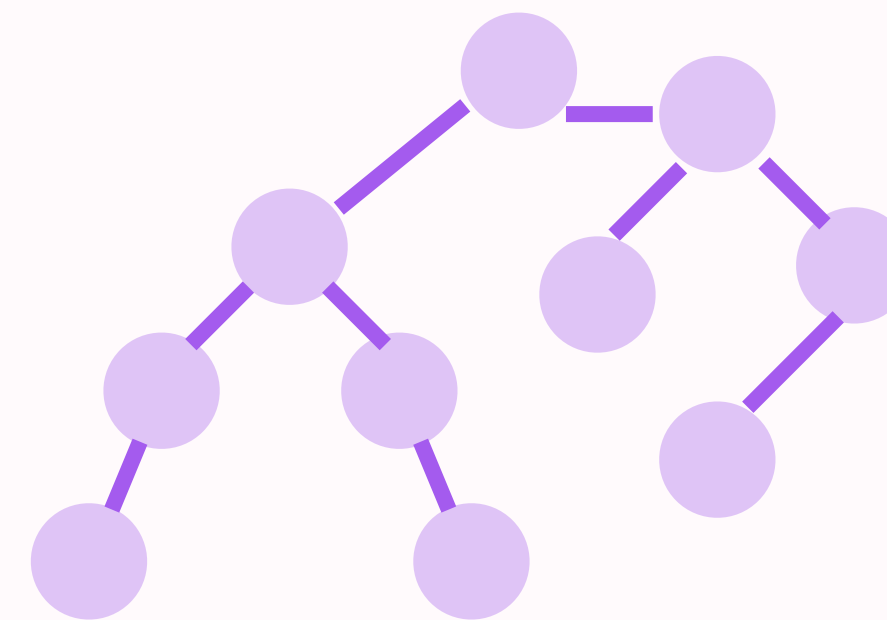
集合

无其他关系



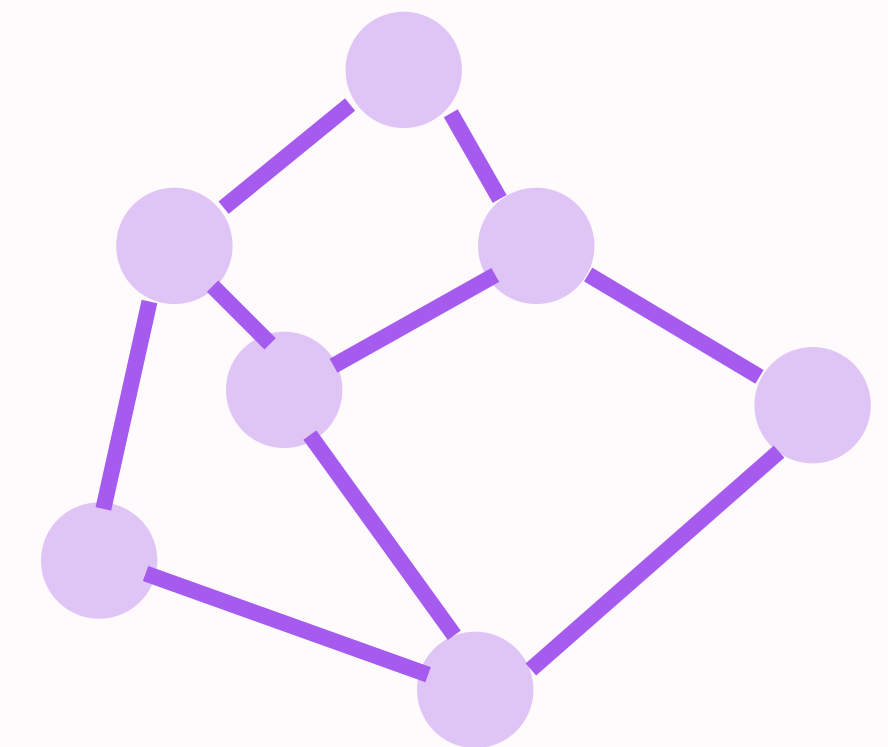
线性结构

一对一



树状结构

一对多



图状或网状结构

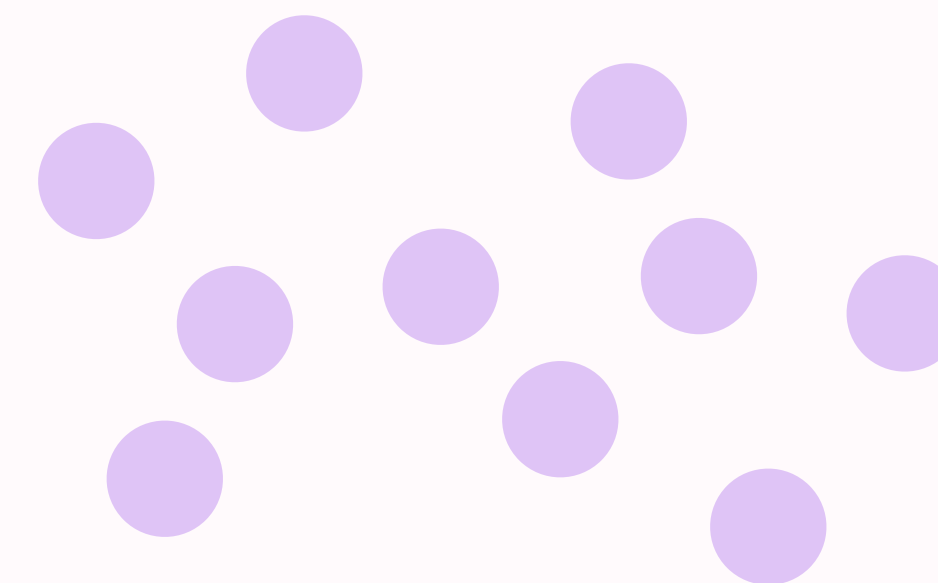
多对多

逻辑结构的四大分类「集合结构实例」

种群分布是一个很好的集合结构实例，

其特点在于：

除了“同属于一个集合”的关系外，别无其他关系

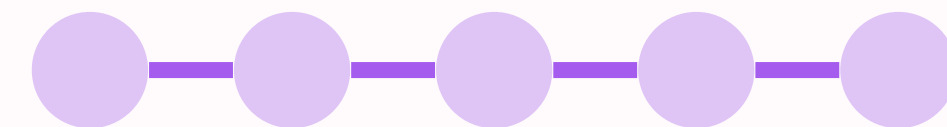


集合

逻辑结构的四大分类「线性结构实例」

传话游戏是一个很好的线性结构实例，

其特点在于：



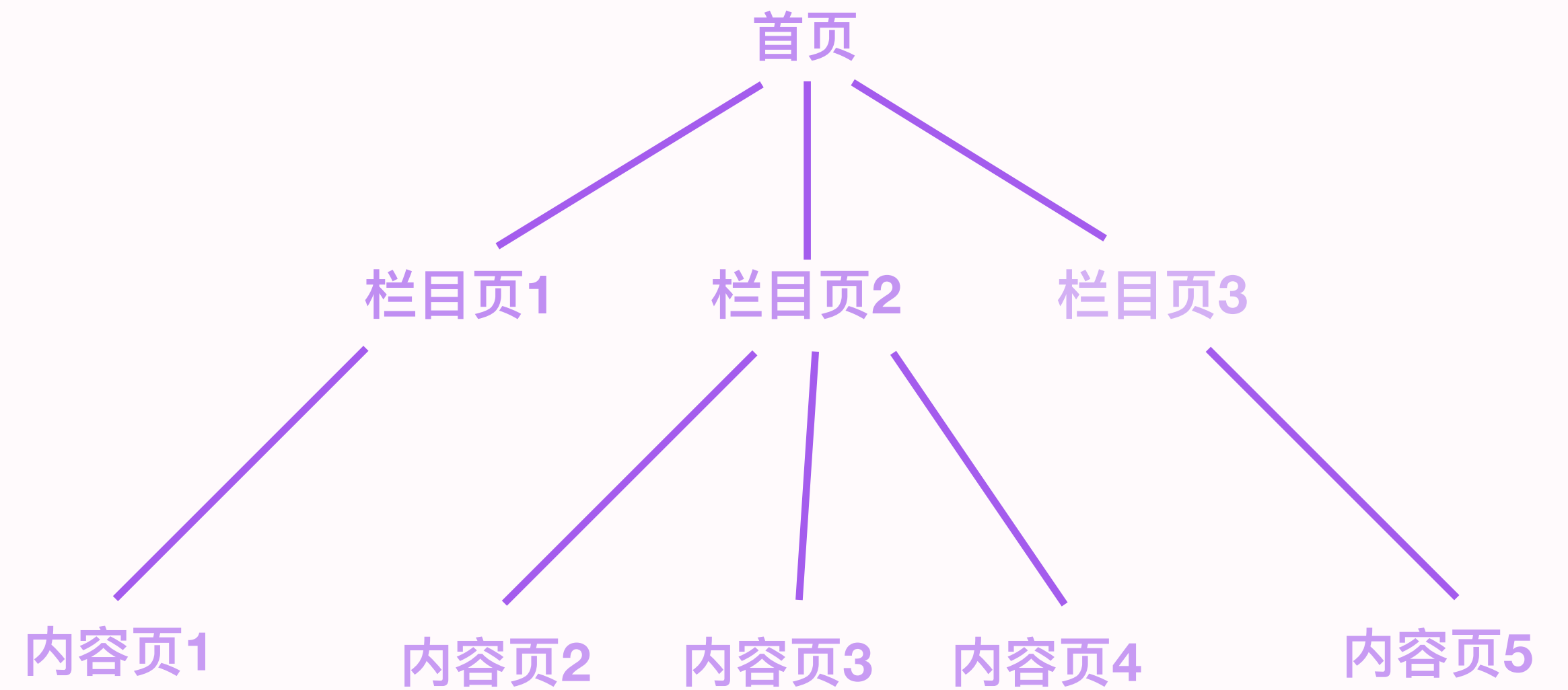
结构中的数据元素之间只存在一对一的关系

线性结构

逻辑结构的四大分类「树状结构实例」

网站结构是一个很好的树状结构实例，
其特点在于：

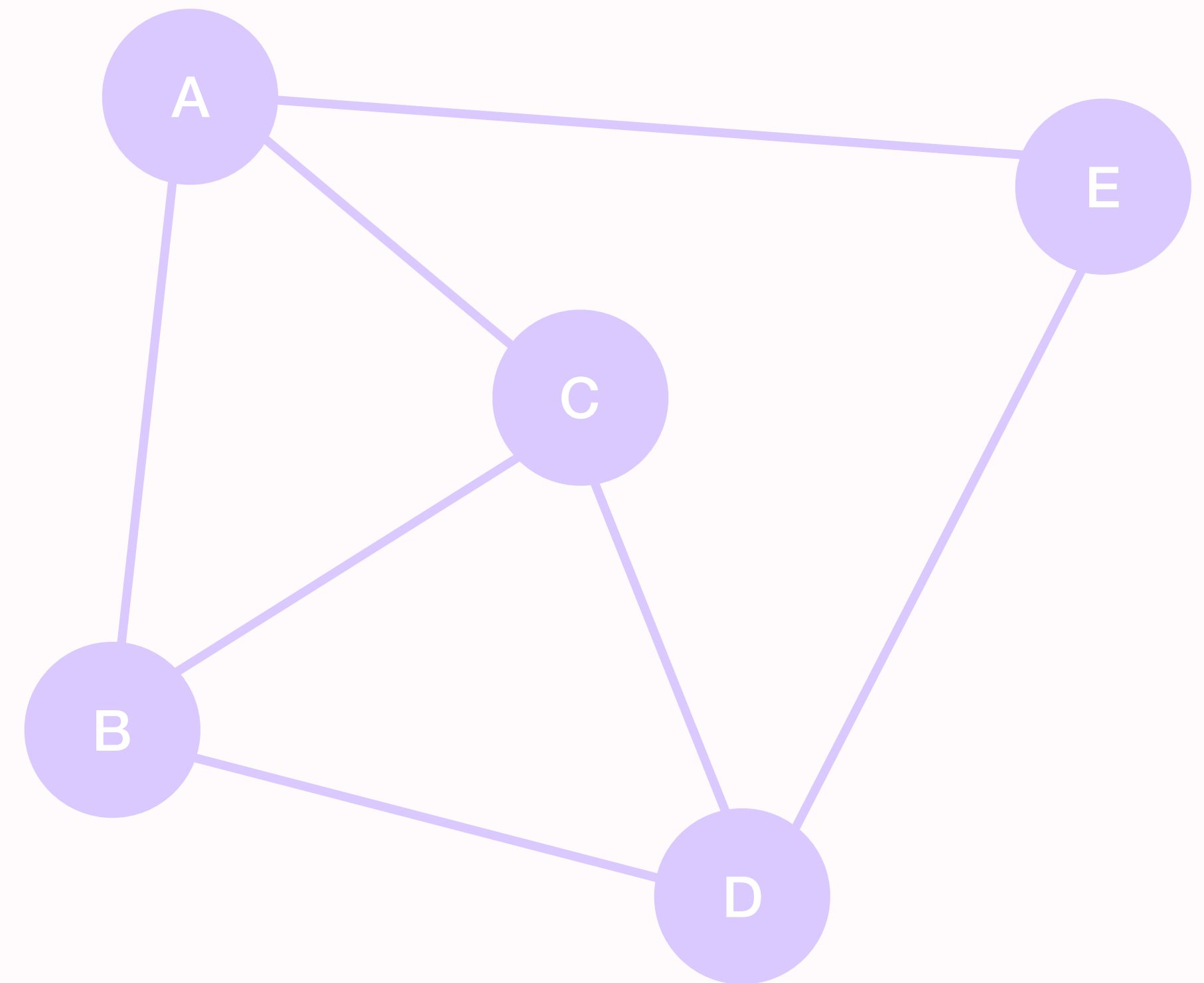
结构中的数据元素之间存在一对多的关系



逻辑结构的四大分类「图状或网状结构实例」

地图是一个很好的图状结构实例，
其特点在于：

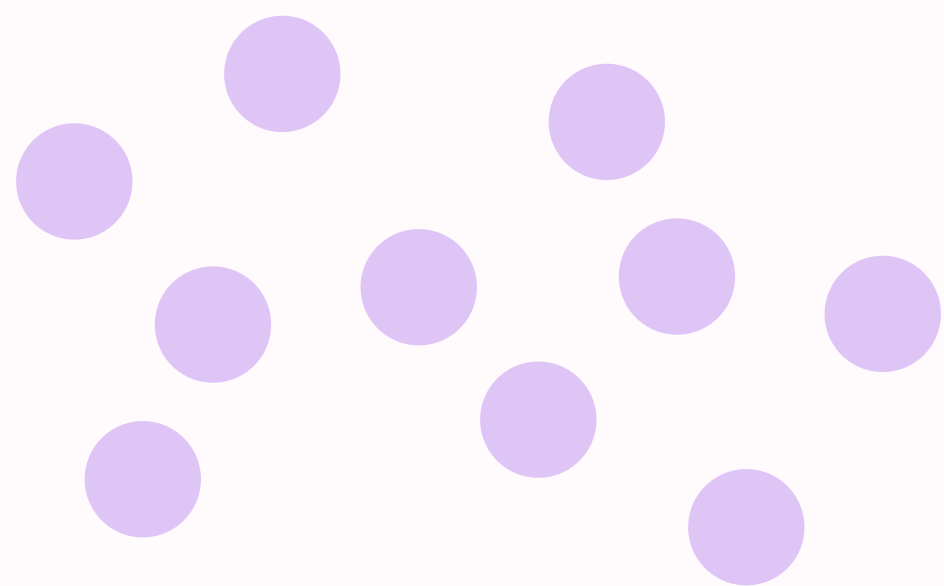
结构中的数据元素之间存在多对多的关系



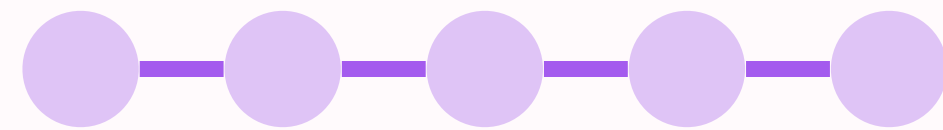
根据二元组判断数据结构的方法

步骤一 / 对于一个已知的二元组，按顺序列出元素之间的关系；

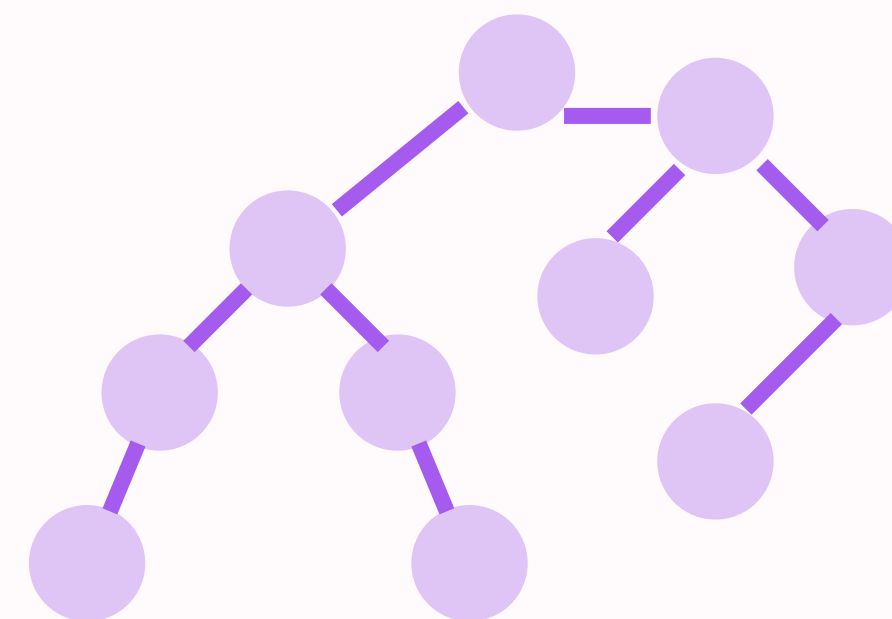
步骤二 / 对照集合、线性结构、树状结构、图状或网状结构，判断二元组对应的数据结构。



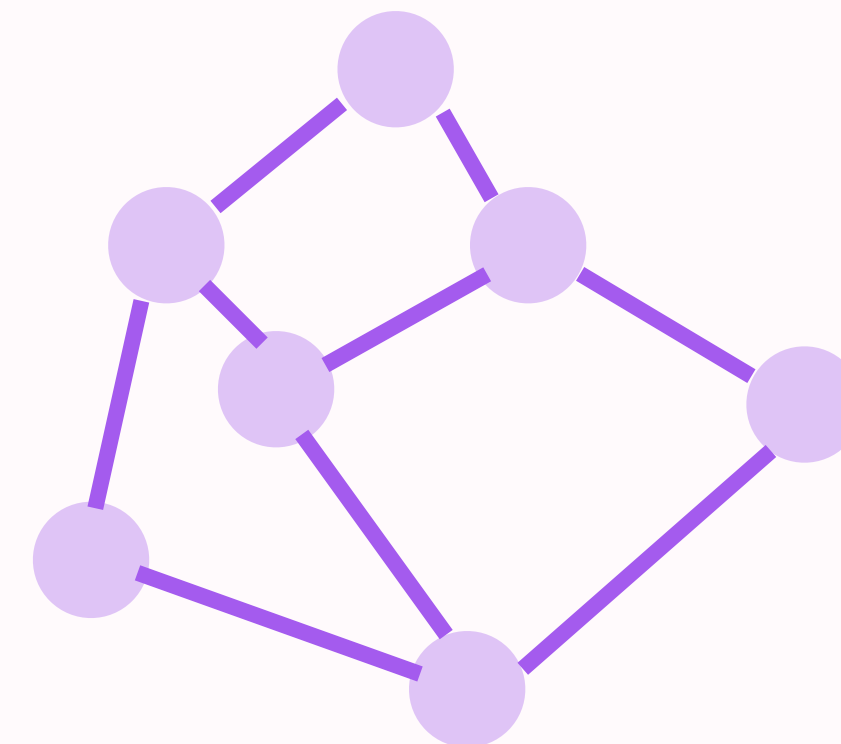
集合



线性结构



树状结构

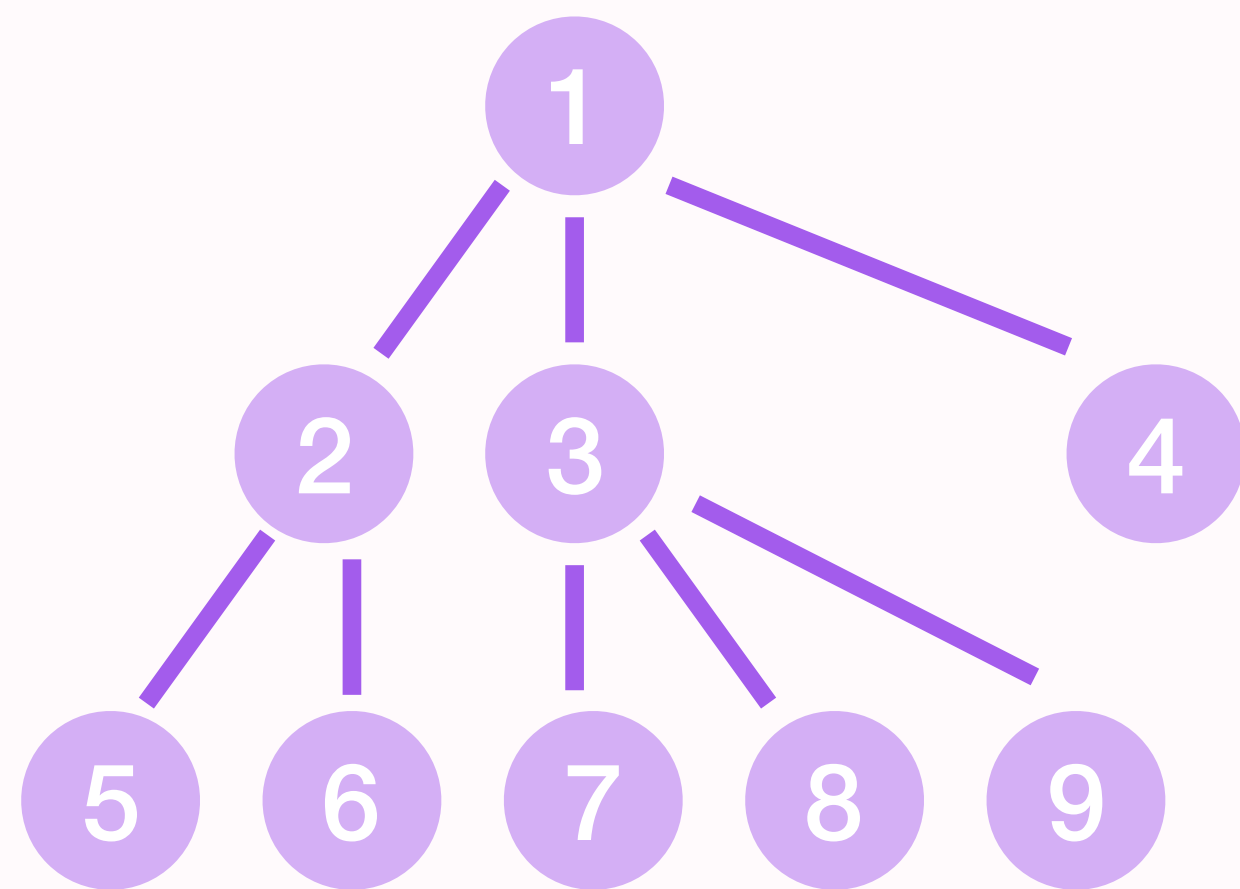


图状或网状结构

例题1-4 / 设某数据结构的二元组形式表示为 $A = (D, R)$,

$D = \{01, 02, 03, 04, 05, 06, 07, 08, 09\}$, $R = \{r\}$, $r = \{<01, 02>, <01, 03>, <01, 04>, <02, 05>, <02, 06>, <03, 07>, <03, 08>, <03, 09>\}$,
则数据结构 A 是何种结构?

解析1-1 / 连接此数据结构得:



即数据结构 A 为树形结构。

存储结构

数据的存储结构又称为**物理结构**。在物理结构表示数据结构中，主要研究数据元素的储存和关系的储存，别称为**数据元素的映象**和**关系的映象**。

物理结构

对于任意一个关系，可分为顺序存储、链式存储、索引存储、散列存储。

	优点	缺点
顺序存储	<div>随机存取</div> <div>每个元素占用最少的存储空间</div>	<div>只能使用相邻的存储单元</div> <div>可能产生较多的外部碎片</div>
链式存储	<div>不会出现碎片现象</div> <div>充分利用所有存储单元</div>	<div>只能顺序存取</div> <div>因存储指针而占用额外的存储空间</div>
索引存储	<div>检索速度快</div>	<div>索引表占用较多存储空间</div> <div>增加和删除数据时修改索引表花费时间</div>
散列存储	<div>检索、增加、删除速度快</div>	<div>可能出现元素存储单元的冲突</div> <div>解决冲突会增加时间和空间开销</div>

例题1-5 / 链式存储设计时，结点内的存储单元地址（ ）

- A. 一定连续
- B. 一定不连续
- C. 不一定连续
- D. 部分连续，部分不连续

解析1-5 / 链式存储设计时，各个不停结点的存储空间可以不连续，但是**结点内**的存储单元地址则**必须连续**

例题1-6

以下关于数据结构的说法中，正确的是（ ）

- A. 数据的逻辑结构独立于其物理结构
- B. 数据的物理结构独立于其逻辑结构
- C. 数据的逻辑结构唯一决定了其物理结构
- D. 数据结构仅由其逻辑结构和物理结构决定

解析1-6

A

- A. 数据的逻辑结构以面向实际问题的角度出发，只采用抽象表达方式，独立于存储结构；
- B. 数据的物理结构是逻辑结构在计算机上的映射，不能独立于逻辑结构存在；
- C. 数据的物理结构有多种选择；
- D. 数据结构包括三要素，缺一不可。

在C语言中的数据类型

小节1 / 数组

小节2 / 指针

小节3 / 结构体

C语言中的数据类型

基本类型	数值类型	整型	short短整型	int整型	long长整型
		浮点型	float单精度型	double双精度型	
	字符类型	char			
	枚举类型	enum			
构造类型	数组				
	结构体	struct			
	共用体	union			
指针类型	指针	*			
	空类型	_Bool			

C语言中的数据类型

基本类型	数值类型	整型 short短整型 int整型 long长整型 浮点型 float单精度型 double双精度型
	字符类型	char
	枚举类型	enum
构造类型	数组	
	结构体	struct
	共用体	union
指针类型	指针	*
	空类型	_Bool

在C语言中的数据类型

小节1 / 数组

小节2 / 指针

小节3 / 结构体

数组

什么是数组？

有序的元素序列。

为什么要用数组？

在运用多个数据存储而又不想定义多个变量时，或不知道数据个数的时候，采用循环输入变量到数组中，输出也用循环输出数组，可以使程序更加方便简洁。

定义：

类型说明符 数组名 [常量表达式];



类型说明符 数组名 [常量表达式];

```
int a[5];
```

```
int a[5]={1,2,3,4,5};
```

```
int a[5]={1,2,3,4,5};
```

```
int a[5]={1,2,3};
```

新建数组的常见范式

```
#include <stdio.h>

int main ()
{
    int n[ 10 ]; // n 是一个包含 10 个整数的数组
    int i,j;

    //初始化数组元素
    for ( i = 0; i < 10; i++ )
    {
        n[ i ] = i; //设置元素 i 为 i
    }

    //输出数组中每个元素的值
    for ( j = 0; j < 10; j++ )
    {
        printf("元素[%d] = %d\n", j, n[j] );
    }

    return 0;
}
```

新建数组的常见范式



```
#include <stdio.h>

int main ()
{
    int n[ 10 ]; // n 是一个包含 10 个整数的数组
    int i,j;

    //初始化数组元素
    for ( i = 0; i < 10; i++ )
    {
        n[ i ] = i; //设置元素 i 为 i
    }

    //输出数组中每个元素的值
    for ( j = 0; j < 10; j++ )
    {
        printf("元素[%d] = %d\n", j, n[j] );
    }

    return 0;
}
```



```
元素[0]=0
元素[1]=1
元素[2]=2
元素[3]=3
元素[4]=4
元素[5]=5
元素[6]=6
元素[7]=7
元素[8]=8
元素[9]=9
```


新建数组的其他可行范式



//第一种形式

```
int [ ] a=new int [5];
```

```
a[0]=10;
```

//第二种形式

```
int [ ] a=new int [ ]{1,2,3};
```

//第三种形式

```
int [ ] i={1,2,3,4};
```

在C语言中的数据类型

小节1 / 数组

小节2 / 指针

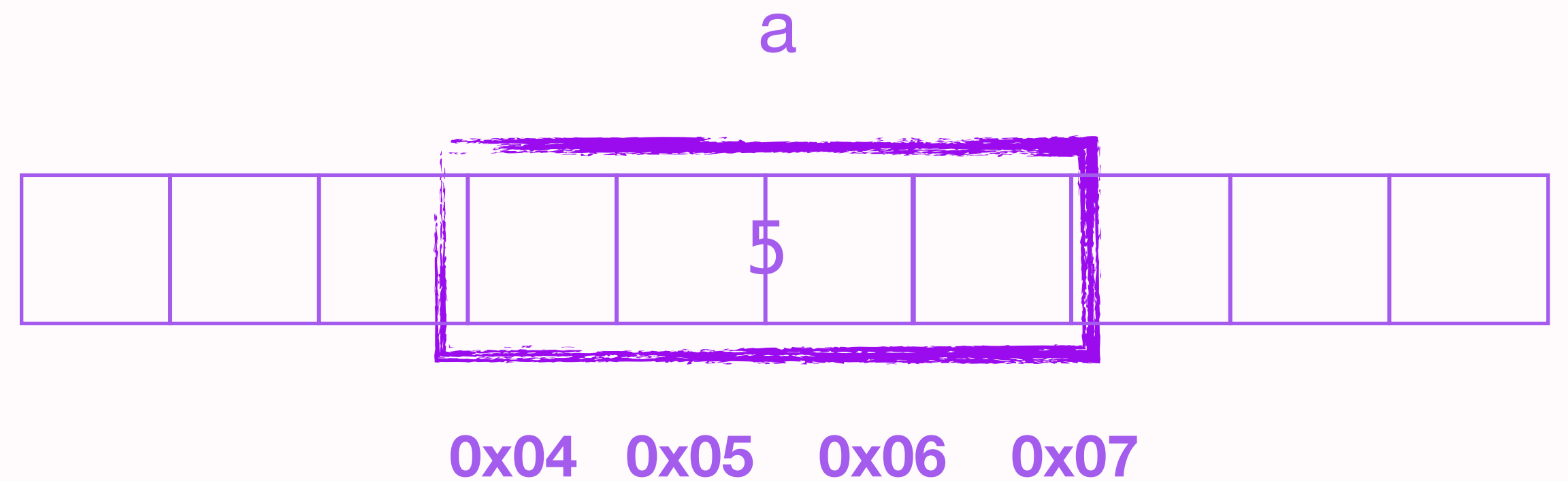
小节3 / 结构体

指针

什么是指针？

一个变量的地址就称为该变量的指针。
指针就是地址，而地址就是内存单元的编号。

内存：



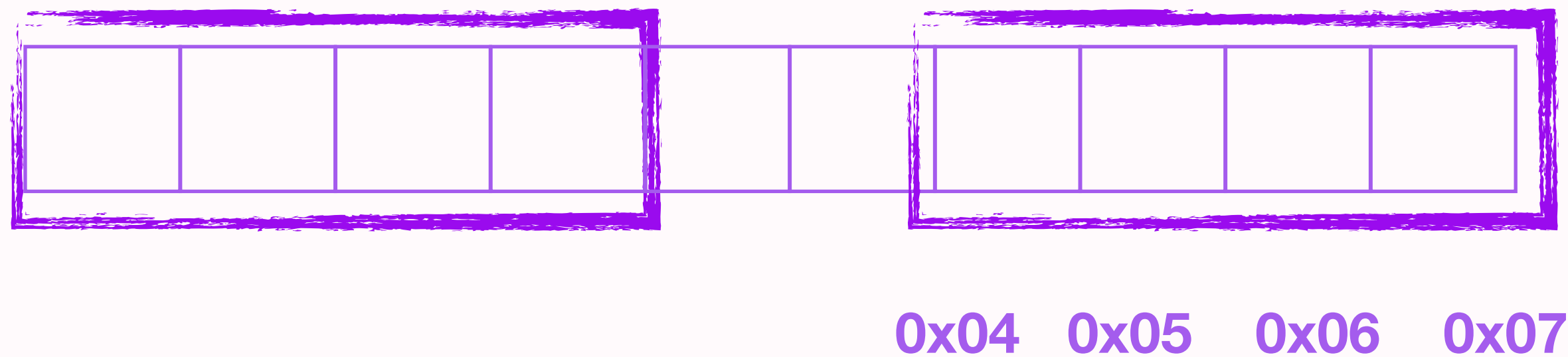
取地址符&

取变量地址的符号。



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

```
int main(void)
{
    int a=5;
    int *p=&a;
    printf( "%d\n",*p);
    return 0;
}
```



取地址符&

取变量地址的符号。



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

```
int main(void)
{
```

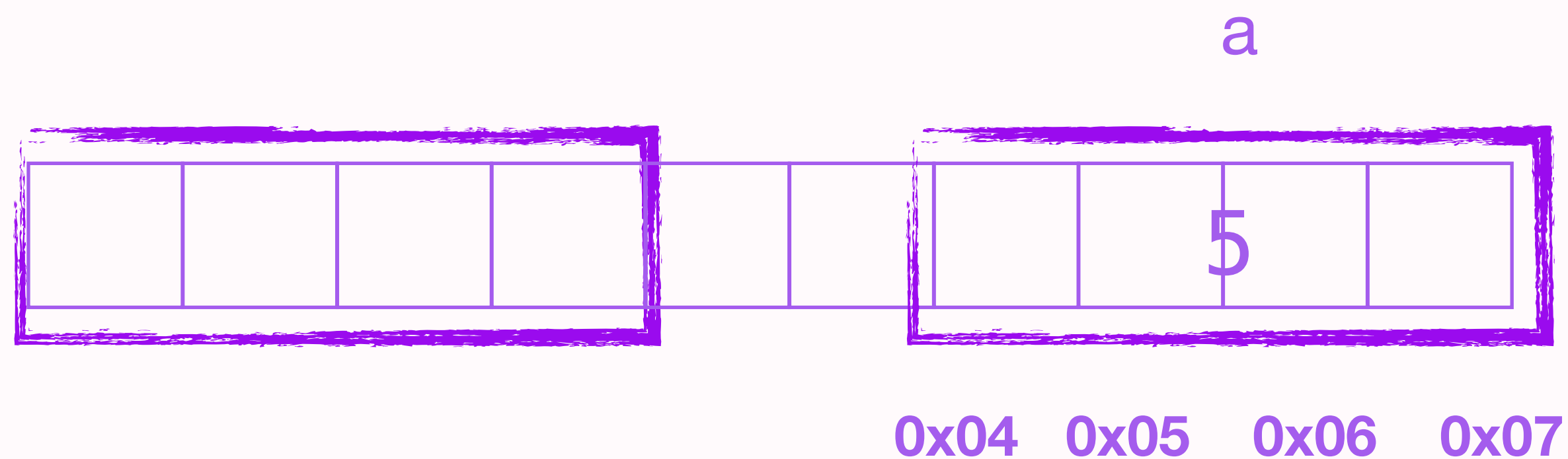
```
int a=5;
```

```
int *p=&a;
```

```
printf( "%d\n",*p);
```

```
return 0;
```

```
}
```



取地址符&

取变量地址的符号。



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

```
int main(void)
{
```

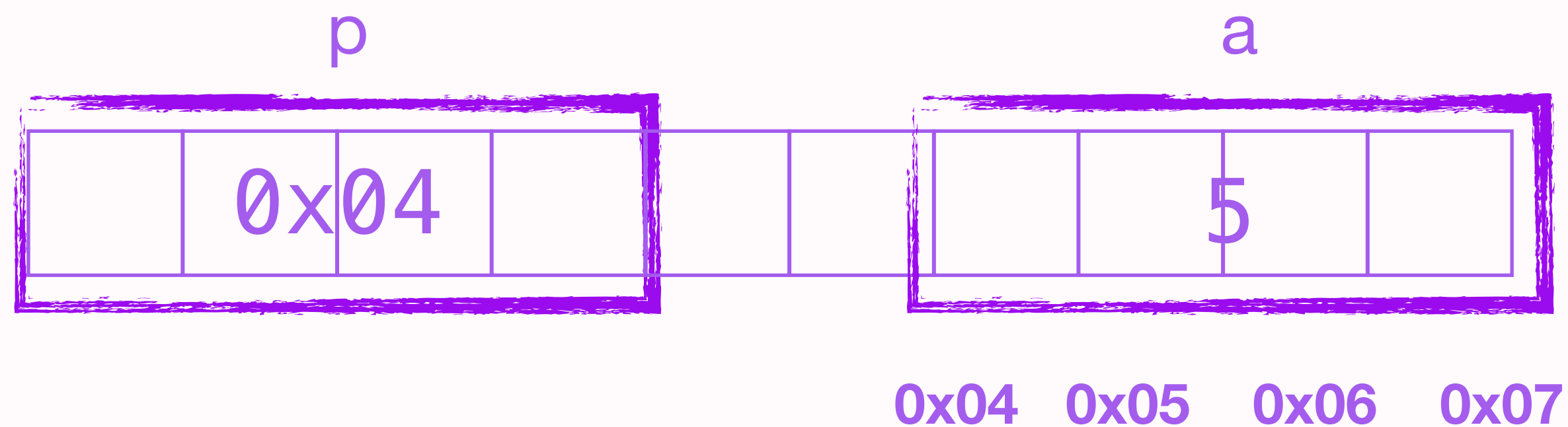
```
int a=5;
```

```
int *p=&a;
```

```
printf( "%d\n", *p);
```

```
return 0;
```

```
}
```



取地址符&

取变量地址的符号。



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

```
int main(void)
{
```

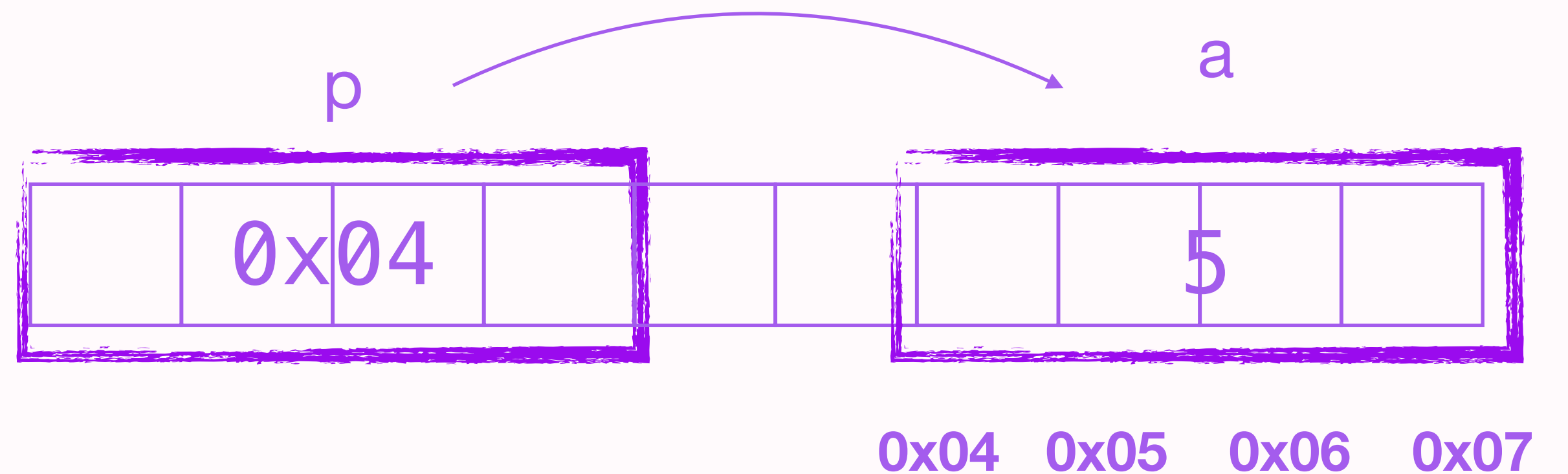
```
int a=5;
```

```
int *p=&a;
```

```
printf( "%d\n", *p);
```

```
return 0;
```

```
}
```



取地址符&

取变量地址的符号。



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

```
int main(void)
{
```

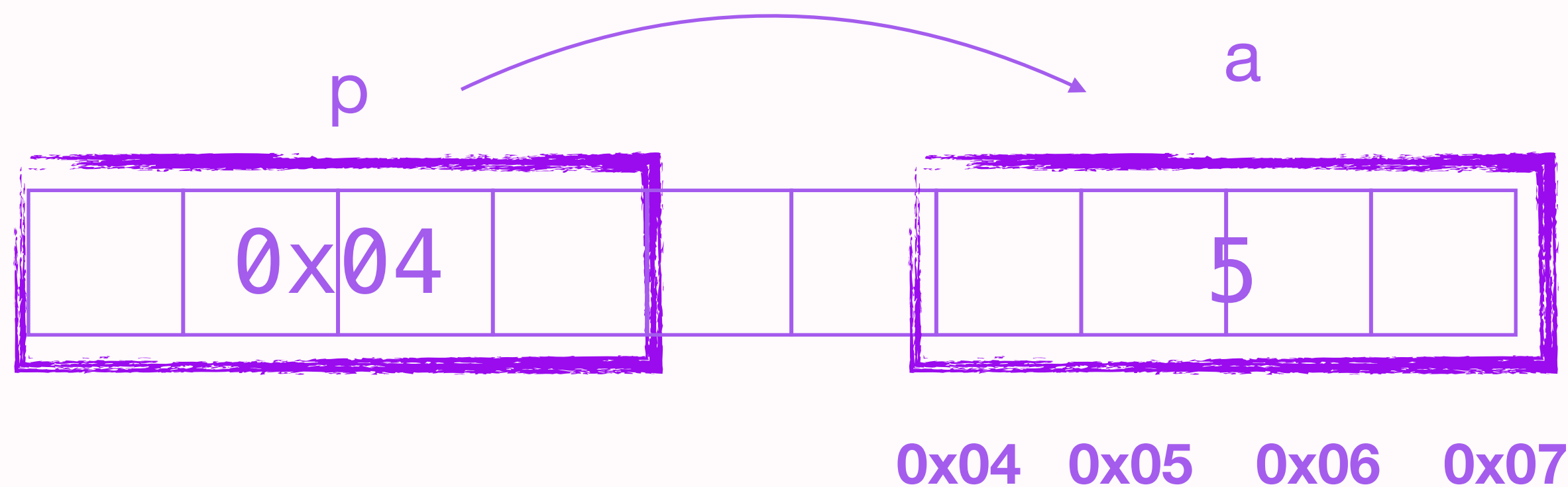
```
int a=5;
```

```
int *p=&a;
```

```
printf("%d\n", *p);
```

a的输出结果

是0x04



指针操作实例

```
#include <stdio.h>

void swap(int a,int b)
{
    int t;
    t=a;
    a=b;
    b=t;
}

int main()
{
    int a=5;
    int b=3;
    swap(a,b);
    printf("a的输出结果是%d\n",a);
    printf("b的输出结果是%d\n",b);
}
```

a的输出结果是5

b的输出结果是3

```
#include <stdio.h>

void swap(int *a,int *b)
{
    int t;
    t=*a;
    *a=*b;
    *b=t;
}

int main()
{
    int a=5;
    int b=3;
    swap(&a,&b);
    printf("a的输出结果是%d\n",a);
    printf("b的输出结果是%d\n",b);
}
```

a的输出结果是3

b的输出结果是5

指针操作实例

swap



main

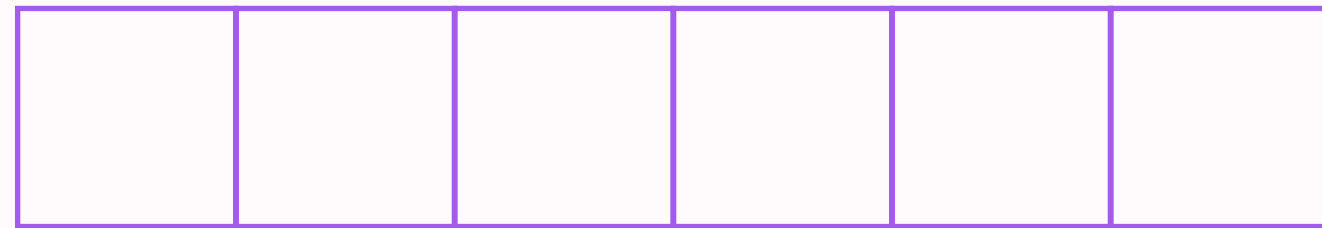


不使用指针

swap



main



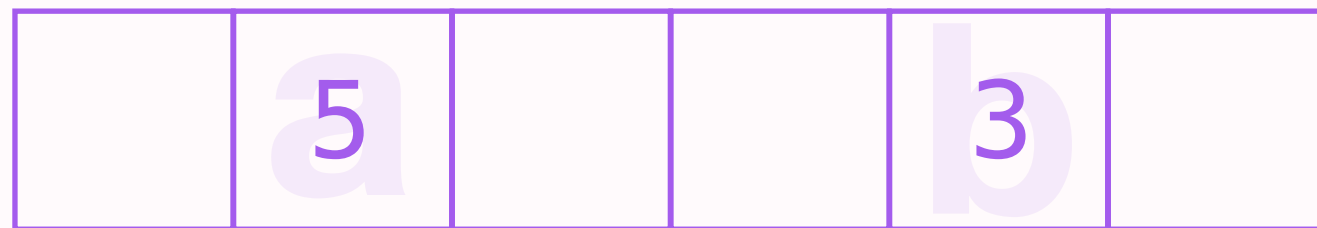
使用指针

指针操作实例

swap



main



不使用指针



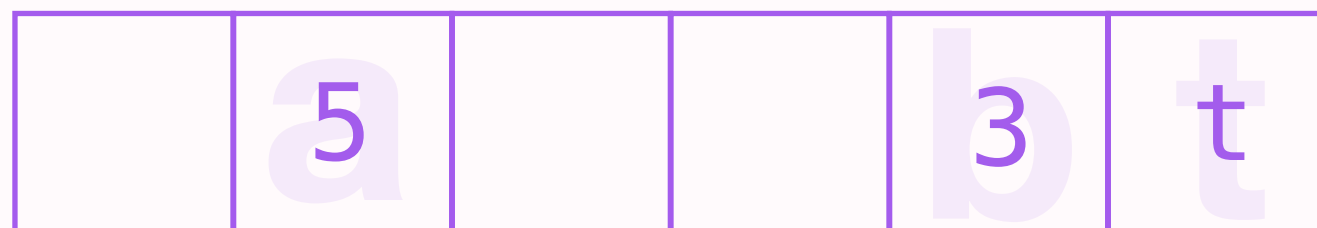
```
#include <stdio.h>

void swap(int a,int b)
{
    int t;
    t=a;
    a=b;
    b=t;
}

int main()
{
    int a=5;
    int b=3;
    swap(a,b);
    printf("a的输出结果是%d\n",a);
    printf("b的输出结果是%d\n",b);
    return 0;
}
```

指针操作实例

swap



main



不使用指针



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

```
void swap(int a,int b)  
{
```

```
    int t;
```

```
    t=a;
```

```
    a=b;
```

```
    b=t;
```

```
}
```

```
int main()  
{
```

```
    int a=5;
```

```
    int b=3;
```

```
    swap(a,b);
```

```
    printf("a的输出结果是%d\n",a);
```

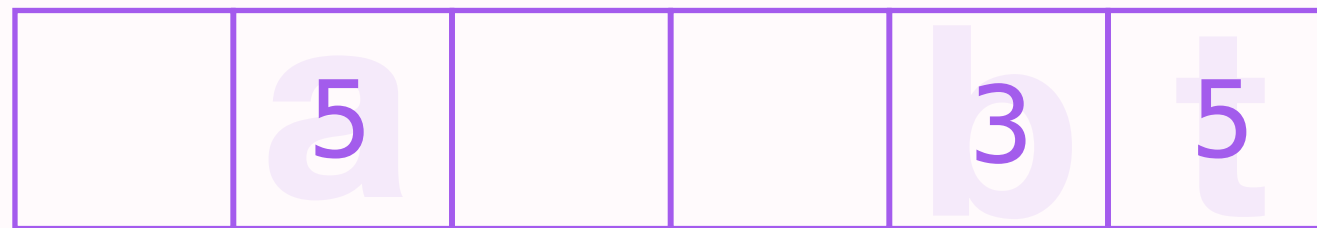
```
    printf("b的输出结果是%d\n",b);
```

```
    return 0;
```

```
}
```

指针操作实例

swap



main



不使用指针



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

```
void swap(int a,int b)
{
```

```
    int t;
```

```
    t=a;
```

```
    a=b;
```

```
    b=t;
```

```
}
```

```
int main()
```

```
{
```

```
    int a=5;
```

```
    int b=3;
```

```
    swap(a,b);
```

```
    printf("a的输出结果是%d\n",a);
```

```
    printf("b的输出结果是%d\n",b);
```

```
    return 0;
```

```
}
```

指针操作实例

swap



main



不使用指针



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

```
void swap(int a,int b)  
{
```

```
    int t;
```

```
    t=a;
```

```
    a=b;
```

```
    b=t;
```

```
}
```

```
int main()  
{
```

```
    int a=5;
```

```
    int b=3;
```

```
    swap(a,b);
```

```
    printf("a的输出结果是%d\n",a);
```

```
    printf("b的输出结果是%d\n",b);
```

```
    return 0;
```

```
}
```

指针操作实例

swap



main



不使用指针



```
#include <stdio.h>

void swap(int a,int b)
{
    int t;
    t=a;
    a=b;
    b=t;
}

int main()
{
    int a=5;
    int b=3;
    swap(a,b);
    printf("a的输出结果是%d\n",a);
    printf("b的输出结果是%d\n",b);
    return 0;
}
```

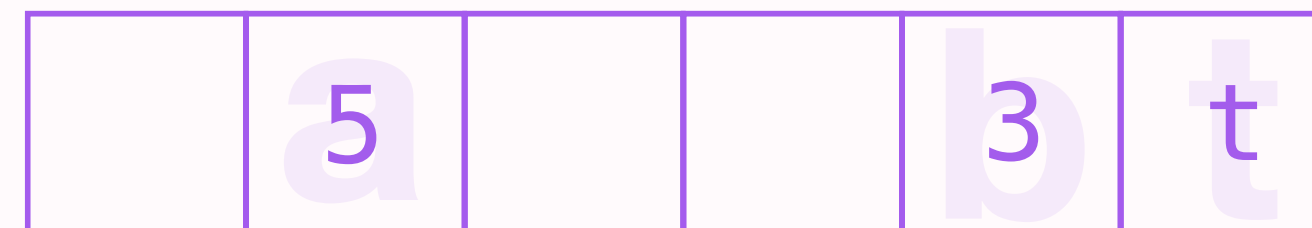
指针操作实例

```
#include <stdio.h>

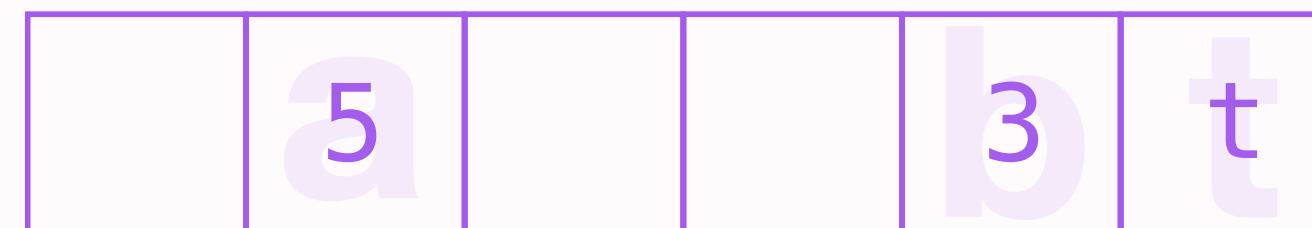
void swap(int *a,int *b)
{
    int t;
    t=*a;
    *a=*b;
    *b=t;
}

int main()
{
    int a=5;
    int b=3;
    swap(&a,&b);
    printf("a的输出结果是%d\n",a);
    printf("b的输出结果是%d\n",b);
    return 0;
}
```

swap



main



使用指针

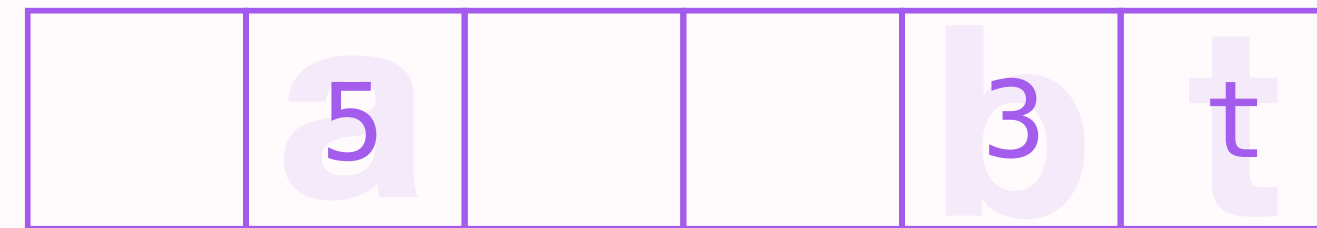
指针操作实例

```
#include <stdio.h>

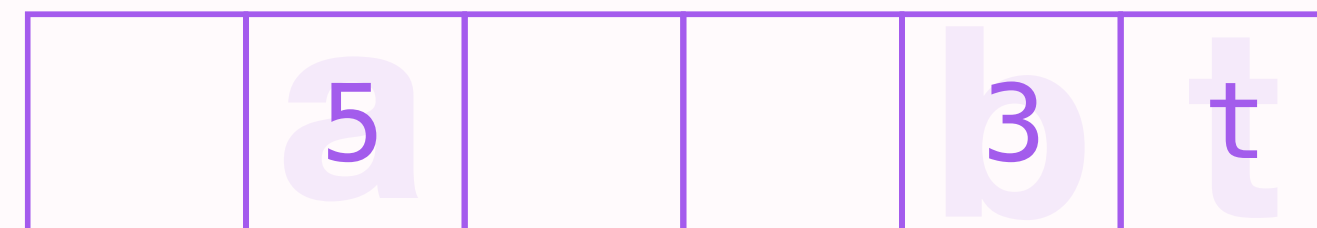
void swap(int *a,int *b)
{
    int t;
    t=*a;
    *a=*b;
    *b=t;
}

int main()
{
    int a=5;
    int b=3;
    swap(&a,&b);
    printf("a的输出结果是%d\n",a);
    printf("b的输出结果是%d\n",b);
    return 0;
}
```

swap



main



使用指针

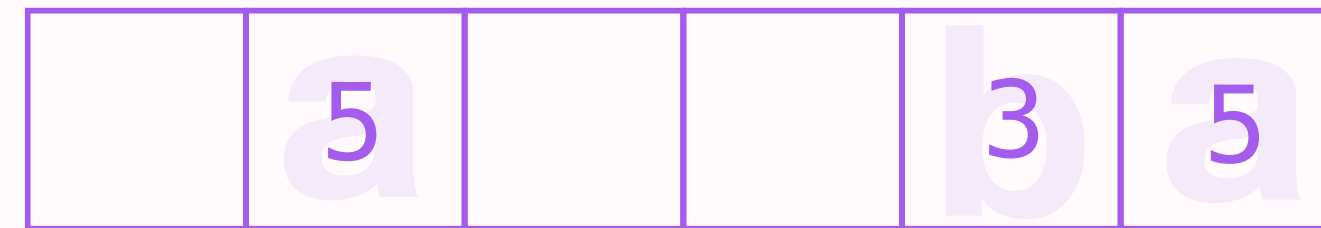
指针操作实例

```
#include <stdio.h>

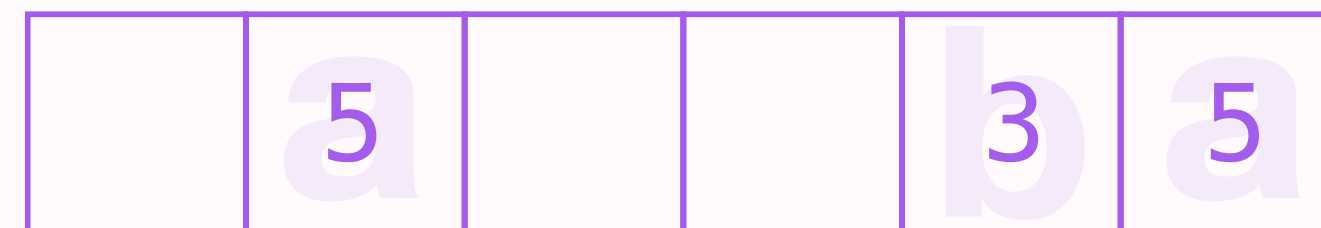
void swap(int *a,int *b)
{
    int t;
    t=*a;
    *a=*b;
    *b=t;
}

int main()
{
    int a=5;
    int b=3;
    swap(&a,&b);
    printf("a的输出结果是%d\n",a);
    printf("b的输出结果是%d\n",b);
    return 0;
}
```

swap



main



使用指针

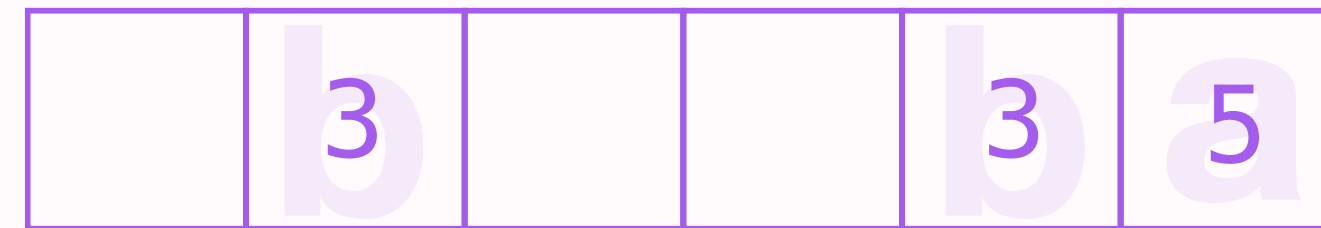
指针操作实例

```
#include <stdio.h>

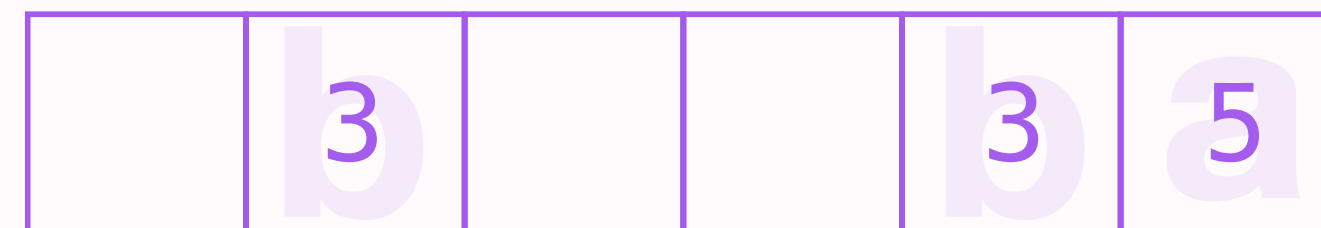
void swap(int *a,int *b)
{
    int t;
    t=*a;
    *a=*b;
    *b=t;
}

int main()
{
    int a=5;
    int b=3;
    swap(&a,&b);
    printf("a的输出结果是%d\n",a);
    printf("b的输出结果是%d\n",b);
    return 0;
}
```

swap



main



使用指针

指针操作实例

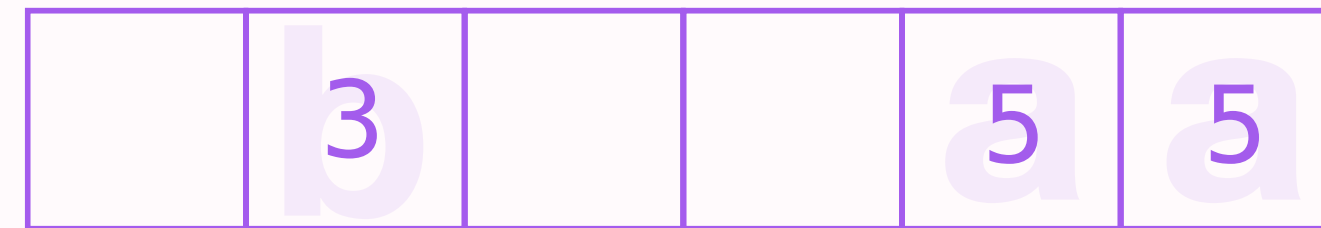


```
#include <stdio.h>
```

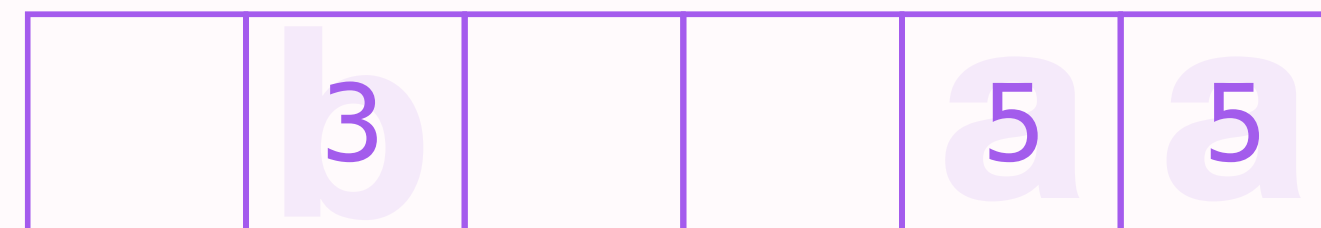
```
void swap(int *a,int *b)
{
    int t;
    t=*a;
    *a=*b;
    *b=t;
}
```

```
int main()
{
    int a=5;
    int b=3;
    swap(&a,&b);
    printf("a的输出结果是%d\n",a);
    printf("b的输出结果是%d\n",b);
    return 0;
}
```

swap

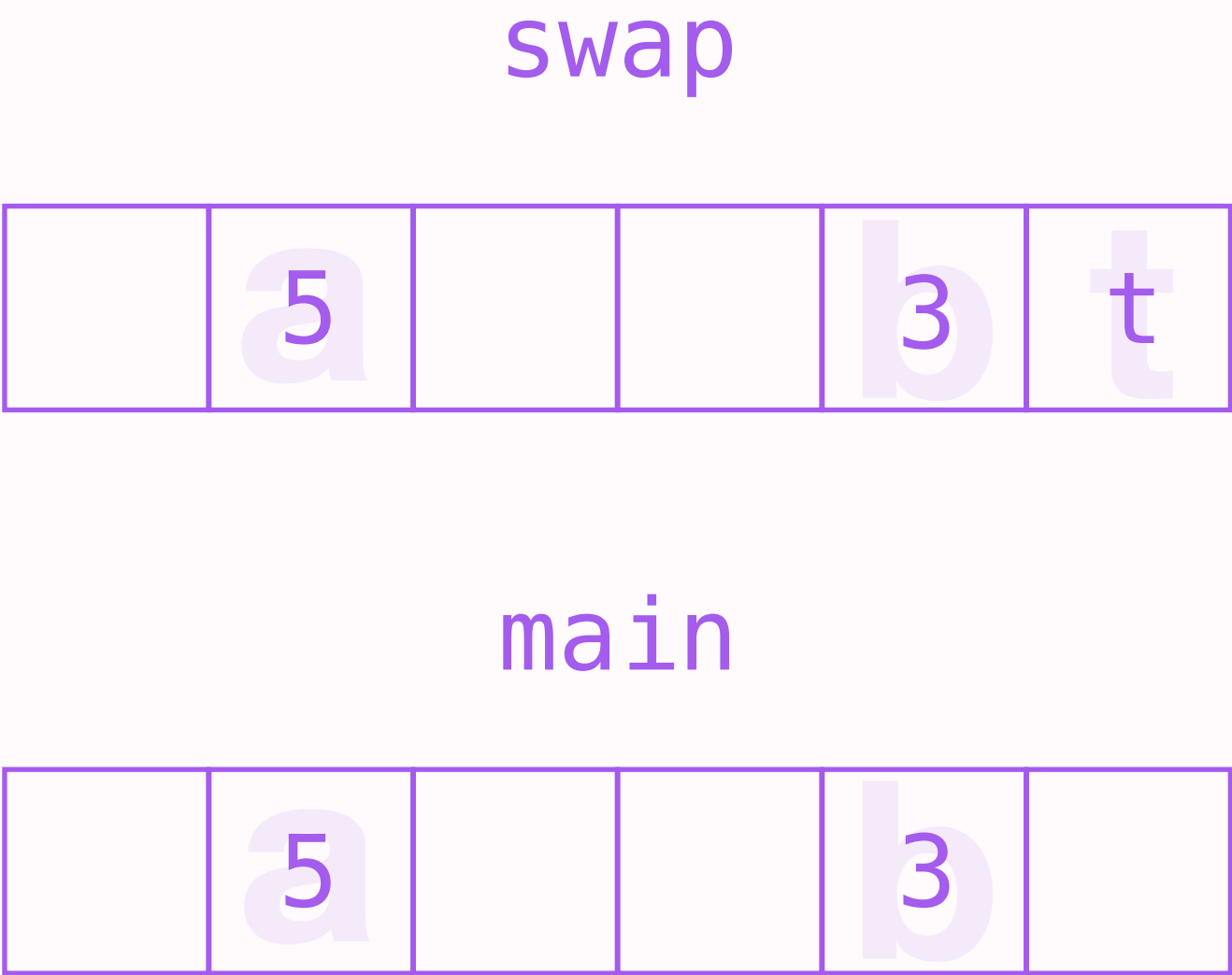


main

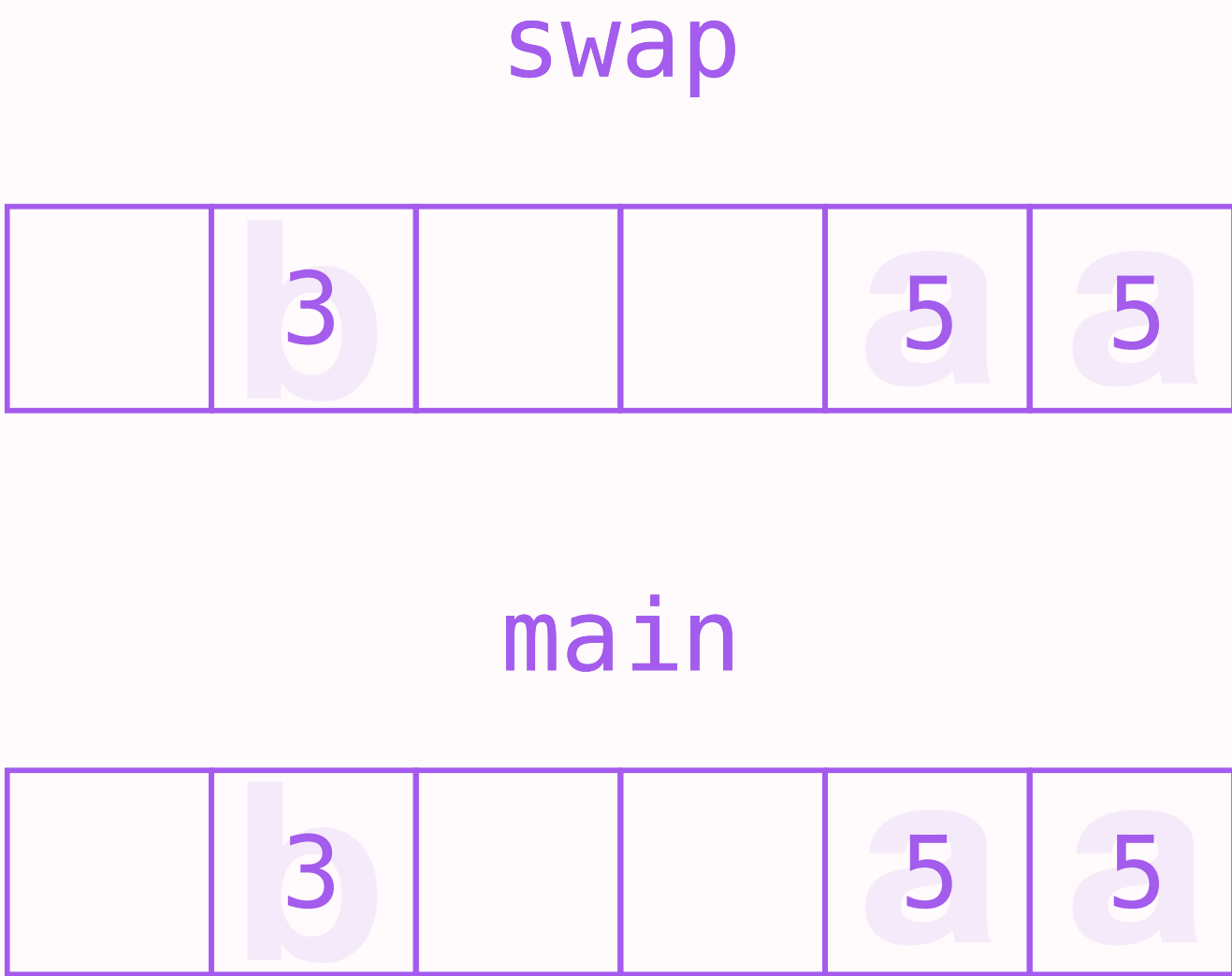


使用指针

指针操作实例



不使用指针



使用指针

在C语言中的数据类型

小节1 / 数组

小节2 / 指针

小节3 / 结构体

结构体

什么是结构体？

结构体就是将不同类型的数据组合合成一个有机的整体，以便于引用。



```
Struct 结构体名  
{  
  成员列表  
};
```

结构体

为什么要用结构体？



```
char name[20];  
int num;  
char sex;  
int age;  
float score;  
char addr[30];
```



```
struct STUDENT  
{  
    char name[20];  
    int num;  
    char sex;  
    int age;  
    float score;  
    char addr[30];  
};
```

在声明结构体类型的时候，不可以对里面的变量进行初始化。

算法与算法分析

小节1 / 算法的基本概念

小节2 / 算法效率的度量

小节3 / 算法的存储空间需求

算法与算法分析

小节1 / 算法的基本概念

小节2 / 算法效率的度量

小节3 / 算法的存储空间需求

算法

对特定问题求解步骤的一种描述。算法是指令的有限序列，其中每一条指令表示一个或多个操作。

算法的五大重要特性

- (1) 有穷性：
- (2) 确定性：
- (3) 可行性：
- (4) 输入：
- (5) 输出：

例题1-7 / 一个算法应该是（ ）。

A.程序

B.问题求解步骤的描述

C.要满足五个基本特性

D.A和C

解析1-7 / **B**

程序不一定要满足有穷性，如死循环、操作系统等，而算法必须有穷。算法代表时间对问题求解步骤的描述，而程序则是算法在计算机上的特定实现。

算法设计的要求：

- (1) 正确性
- (2) 可读性
- (3) 健壮性：
- (4) 效率与存储量需求

算法与算法分析

小节1 / 算法的基本概念

小节2 / 算法效率的度量

小节3 / 算法的存储空间需求

算法效率的度量：

时间复杂度：

一个语句的频度是指该语句在算法被重复执行的次数。

$$T(n) = O(f(n))$$

算法效率的度量：

空间复杂度：

一个语句的频度是指该语句在算法被重复执行的次数。

$$S(n) = O(g(n))$$

例题1-8 / 下面算法的时间复杂度是_____。



```
for( i=1, i<=n; ++i )
    for( j=1; j<=n; ++j )
    {
        c[i][j]=0;
        for( k=1; k<=n; ++k )
            c[i][j] += a[i][k] * b[k][j];
    }
```

解析1-8 / 由于是一个三重循环，每个循环从1到n，则总次数为： $n \times n \times n = n^3$
时间复杂度为 $T(n) = O(n^3)$

例题1-9 / 下面算法的时间复杂度是_____。



```
for( i=2; i<=n; ++i )  
    for( j=2; j<=i-1; ++j )  
    {  
        ++x;  
        a[ i, j ]=x;  
    }
```

解析1-9 / 语句频度为：

$$\begin{aligned} 1+2+3+\dots+n-2 &= (1+n-2) \times (n-2)/2 \\ &= (n-1)(n-2)/2 \\ &= n^2-3n+2 \end{aligned}$$

所以，时间复杂度为 $O(n^2)$ ，即此算法的时间复杂度为平方阶。

例题1-10/ 下面算法的时间复杂度是_____。



```
y=0;  
while( n>=( y+1)*( y+1) )  
{  
    y++;  
}
```

解析1-10/ $y++$ 的次数恰好与 $T(n)$ 成正比，则记 t 为该程序的执行次数并令 $t=y-0$ ，即 $y=t$ ， $(t+1) * (t+1) < n$ ，得出 $t < \sqrt{n}-1$

即 $T(n) = O(\sqrt{n})$

例题1-11 / 下面算法的时间复杂度是_____。



```
int i=1;  
while( i<=n )  
    i=i*2;
```

解析1-11 / $i \times 2$ 的次数正是执行次数 t ，因此有 $2^t \leq n$ ，取对数后，即得 $t \leq \log_2 n$ ，则 $T(n) = (\log_2 n)$

例题1-12/ 下面算法的时间复杂度是_____。



```
for ( i=n-1; i>1; i-- )
    for ( j=1; j<i; j++ )
        if ( A[j]>A[j+1] )
            A[j]<-->A[j+1];
```

解析1-12/ 有的情况下，算法中基本操作重复执行的次数还随问题的输入数据集不同而不同。

当所有相邻元素都为顺序时是最好情况：0次；而当都为逆序时，最坏情况： $(n-2)(n-1)/2=O(n^2)$

最坏情况下的时间复杂度为： $O(n^2)$

例题1-13/ 已知两个 长度分别为m和n的升序链表，若将它们合并为一个m+n的降序链表，则最坏情况下的时间复杂度是（ ）

A. $O(n)$

B. $O(m*n)$

C. $\min(m,n)$

D. $\max(m,n)$

解析1-13/ **D**

两个升序链表合并，两两比较表中元素，每比较一次确定一个元素的链接位置，取较小元素（头插法）。当一个链表比较结束后，将另一个链表的剩余元素插入即可。

最坏的情况是两个链表中的元素依次进行比较，时间复杂度为 $O(\max(m,n))$ 。

六种常用算法时间复杂度的大小比较

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3)$$

指数时间的关系为：

$$O(2^n) < O(n!) < O(n^n)$$

算法与算法分析

小节1 / 算法的基本概念

小节2 / 算法效率的度量

小节3 / 算法的存储空间需求

算法的存储空间需求

算法的空间复杂度定义为： $S(n) = O(g(n))$

表示随着问题规模 n 的增大，算法运行所需存储量的增长率与 $g(n)$ 的增长率相同。

算法的存储量包括

1.输入数据所占空间。

2.程序本身所占空间。

3.辅助变量所占空间。

算法的其他术语

若输入数据所占空间只取决于问题本身，和算法无关，则只需要分析除输入和程序之外的辅助变量所占的额外空间。

若所需额外空间相对于输入数据量来说是常数，则称此算法为原地工作。

若所需存储量依赖于特定的输入，则通常按最坏情况考虑。

例题1-14 / 判断题：算法原地工作的含义是指不需要任何额外的辅助空间

解析1-14 / 错误，算法原地工作是指算法所需的辅助空间是常量。

例题1-15/ 判断题：在相同的规模 n 下，复杂度 $O(n)$ 的算法在时间上总是优于复杂度 $O(2^n)$ 的算法

解析1-15/ 对，题中是指算法的时间复杂度，不要想当然认为是程序（该算法的实现）的具体执行时间而赋予 n 一个特殊的值。时间复杂度为 $O(n)$ 的算法，必然总是优于时间复杂度为 $O(2^n)$ 的算法。

例题1-16 / 判断题：所谓时间复杂度是指最坏情况下，估算算法执行时间的一个上界

解析1-16 / 正确，时间复杂度总是考虑最坏情况下的时间复杂度，以保证算法的运行时间不会比它更长。

例题1-17/

判断题： 同一个算法，实现语言的级别越高，执行效率就越低

解析1-17/

正确，这句话为严蔚敏所著教材中的原话，该问题在论坛讨论过多年，对于这种在语言层次的效率问题，建议不要以特例程序来解释其优劣，该结论肯定没有任何错误的。

数据结构绪论

斐多课堂  数据结构  第一讲
Phaedo Classes