回顾先前问题：x值的来源

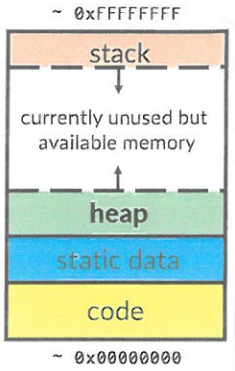
```
#include <stdio.h>
#include <stdlib.h>

void undefined_local() {
  int x;
  printf("x = %d\n", x);
}

void some_calc(int a) {
  a = a % 2 ? rand() : -a;
}

int main(void) {
  for (int i = 0; i < 5; i++) {
    some_calc(i * i);
    undefined_local();
  }
  return 0;
}
```

栈的分配，数据的再使用

② 栈的分配

// → ① 函数调用

// → ③ 函数调用



Heap is dynamic data section

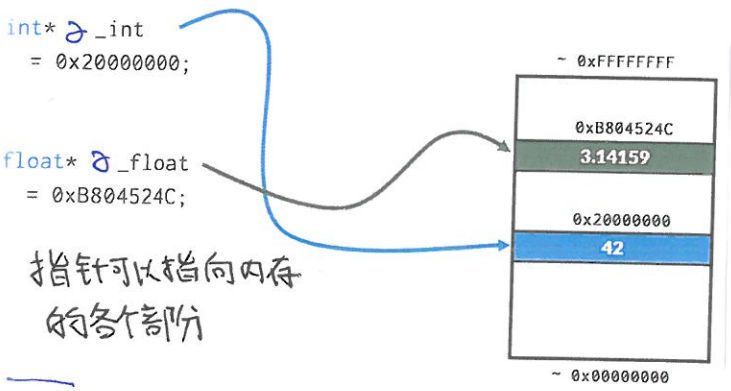
管理该内存很复杂

```
#include <stdlib.h>  // 用于内存分配

void main(void) {
  // 数组要10个integer.
  // malloc返回heap中的分地址
  int* data = malloc(sizeof(int) * 10);
}
```

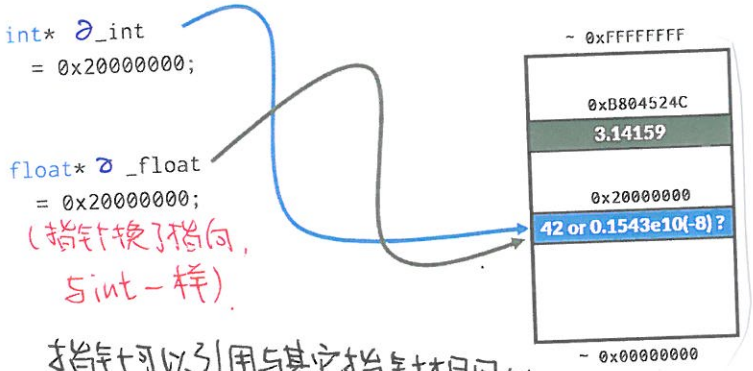
指针：保存内存地址的特定变量类型

可以创建指向内存中任何地址的指针，告知CPU将内存解释为什么类型的数据：*放在末尾

如：int* my_integer_somewhere



指针可以指向内存的各个部分



指针可以引用与其它指针相同的地址。

一但若上，之前分配的数据不允访问。

解除引用指针

一如果我们有一个保存地址的变量，

正常操作会改变地址，而不是指针引用的值。

dereference operator *

```
int* a_data = 0x00800000; // 这个地址任意，无特殊性
a_data = 0xffffffff;      // 重分配地址
*a_data = 42;             // 重分配值

int data = 0xc0de;        // 初始化一个新变量
data = *a_data;           // 从指针中取值
```

另：C隐式转换给予的value

一若错误地给地址取值，仍能编译，但会收到警告。

```
int* a_data = 0x00800000; // 一个任意地址

int* another_a_data = _data; // 分配地址
int* a_data = *a_data;       // 把值给地址
```

会出现编译警告：从int初始化int*

使指针指向integer而没有casting

称指针为数据的"reference". reference operator (&)

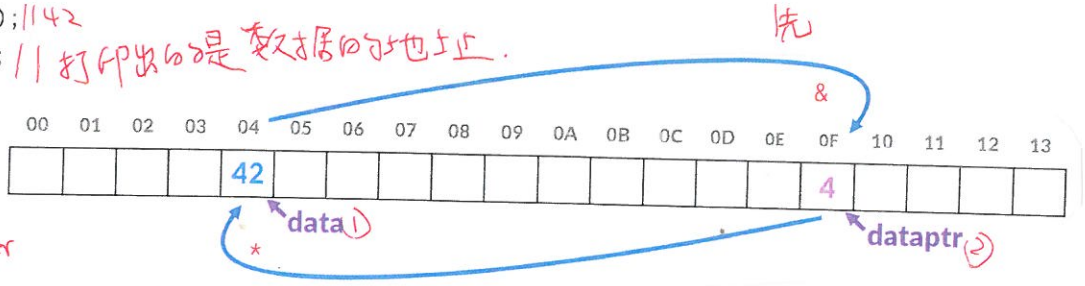
```
int* ∂_data = 0x00800000; // 任意位置

int data = 0xc0de;          // 初始化data变量
∂_data = &data; ← reference   // 把data地址分给指针
*∂_data = 42; ← dereference   // 把value 42放入内存
printf("%d\n", data);
```

output: 42

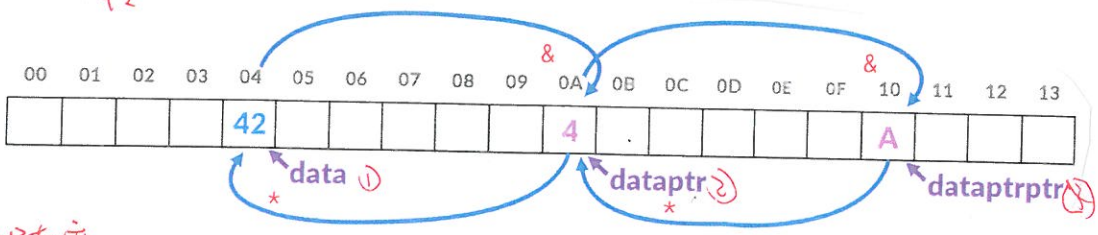
```
int data = 42;
int* ☞_data = &data; //

//
int** ☞_☞_data = &☞_data; // =>

//
*(*☞_☞_data) = -64; //
```

```
int data = 42;
int* dataptr = &data; // 存储数据的内地址

// 指向int的pointer
int** dataptrptr = &dataptr; // 存储 dataptr 的地址

// 解引用
*(*dataptrptr) = -64; // 把值存进 data
```

① int data = 42; // 初始化一个新变量
② int* dataptr = &data; // 把data地址分给指针
printf("%d\n", data); // 42
printf("%d\n", *dataptr); // 42
printf("%p\n", dataptr); // 打印出的是数据的内地址.



① int data = 42; datapointer
② int* dataptr = &data; // 把地址分给指针
③ int** dataptrptr = &dataptr; // 把指针变量的地址分给 ptrptr.
printf("%p\n", dataptr);
printf("%p\n", dataptrptr); // & 打印指针的内地址.
printf("%d\n", **dataptrptr);
42



Array 数组: 连续的内存跨度.

在栈上声明数组

```
void main(void) {
  int array[5]; // 5个integer, 会有garbage
}
```

在栈上初始化

```
void main(void) {
  int array[5] = {1, 42, -3};
}
```

在heap上声明数组.

```
void main(void) {

  int* array = (int*)malloc(sizeof(int) * 5);
}
```

在heap上初始化

```
void main(void) {

  int* array = (int*)calloc(5, sizeof(int));
}
```

JAVA PROGRAMMING

Chapter 1 概述

CPU < 控制单元 (control unit)：控调其他组件
逻辑单元 (logic unit)：数值运算 (+ - x÷)；逻辑运算 (比较)

1 byte = 8 bit (二进制数，0或1)

内存 (RAM，随机访问存储器)：由有序字节序列组成
↓
每个字节都有唯一的地址.

汇编语言 (assembly language)：汇编语言 → 汇编器 (assembler) → 机器语言 (二进制)

高级编程语言：(同) 源代码 → 解释器 → 输出
(异) 源代码 → 编译器 → 机器语言 → 执行器 → 输出

操作系统 - 调度
- multiprogramming → 多程序共享CPU
- multithreading → 单程序执行多个任务
- multiprocessing → 多处理器执行单个任务.

API：库，application program interface (应用程序接口)
为开发Java预定义的类和接口

- JDK (Java开发工具包)：开发环境
  - IDE (集成开发环境)：最普通，如Eclipse → 啥都有
  - JRE (Java运行环境)：只提供运行环境，不包含开发工具
- JVM (虚拟机)：用于解释 class 文件.

我们现在写代码的过程：用整合的IDE写出个 xx.java，送去编译，变成 xx.class 的机器语言，交给JVM虚拟机执行.