C语言理论知识总览

Week 17

陈鸿峥

December, 2018

1

简介

简介

- 一些有用的网站
  - 题库 https://www.sanfoundry.com/c-interview-questions-answers/
  - 在线测试 https://rank.sanfoundry.com/c-programming-tests/
  - 问题 https://stackoverflow.com/
  - 参考手册 https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html
  - 圣书 K&R Brian Kernighan and Dennis Ritchie, *The C Programming Language*
- 不会将C的所有理论知识点都讲完，只会选取一些重点、易忽视的点讲

2

为什么要学C

TIOBE Ranking

Dec 2018 Ranking

Dec 2018	Dec 2017	Change	Programming Language	Ratings	Change
1	1		Java	15.932%	+2.66%
2	2		C	14.282%	+4.12%
3	4	▲	Python	8.376%	+4.60%
4	3	▼	C++	7.562%	+2.84%
5	7	▲	Visual Basic .NET	7.127%	+4.66%
6	5	▼	C#	3.455%	+0.63%
7	6	▼	JavaScript	3.063%	+0.59%
8	9	▲	PHP	2.442%	+0.85%
9	-	⇈	SQL	2.184%	+2.18%
10	12	▲	Objective-C	1.477%	-0.02%

Why C Programming Language

- 速度、稳定性、可移植性
- 操作系统、嵌入式系统、驱动程序、底层操作
- 其他语言的编译器、库、解释器(Python、PHP、Perl)
- 科学计算(Mathematica、MATLAB)

Why C Programming Language

- 速度、稳定性、可移植性
- 操作系统、嵌入式系统、驱动程序、底层操作
- 其他语言的编译器、库、解释器(Python、PHP、Perl)
- 科学计算(Mathematica、MATLAB)

然而

C语言并不是必须的，**C++**应用范围更广也会更好用

Why C Programming Language

- 速度、稳定性、可移植性
- 操作系统、嵌入式系统、驱动程序、底层操作
- 其他语言的编译器、库、解释器(Python、PHP、Perl)
- 科学计算(Mathematica、MATLAB)

然而

C语言并不是必须的，**C++**应用范围更广也会更好用

然然而

- 学C++重点在于弄清楚如何面向对象编程(OOP)

Why C Programming Language

- 速度、稳定性、可移植性
- 操作系统、嵌入式系统、驱动程序、底层操作
- 其他语言的编译器、库、解释器(Python、PHP、Perl)
- 科学计算(Mathematica、MATLAB)

然而

C语言并不是必须的，**C++**应用范围更广也会更好用

然然而

- 学C++重点在于弄清楚如何面向对象编程(OOP)
- 千万不要将程序写得跟C一样（不是C with STL）

Why C Programming Language

- 速度、稳定性、可移植性
- 操作系统、嵌入式系统、驱动程序、底层操作
- 其他语言的编译器、库、解释器(Python、PHP、Perl)
- 科学计算(Mathematica、MATLAB)

然而

C语言并不是必须的，**C++**应用范围更广也会更好用

然然而

- 学C++重点在于弄清楚如何面向对象编程(OOP)
- 千万不要将程序写得跟C一样（不是C with STL）
- 生活不只有ACM

3

知识点

3.1

计算机的基础知识

编译的四个阶段


源程序 hello.c
预处理器(Preprocessor)
添加了宏的源程序 hello.i
编译器(Compiler)
汇编程序 hello.s
汇编器(Assembler)
可重定位目标程序 hello.o(bj) + printf.o
链接器(Linker)
可执行二进制程序 hello

一些易混淆的地方

- 解释器(Interpreter)：直接执行高级语言(Python)

一些易混淆的地方

- 解释器(Interpreter)：直接执行高级语言(Python)
- 另外的编译体系(Java Virtual Machine, JVM)
  - 字节码/中间码(byte code)：包含执行程序，由一序列指令代码和数据组成的二进制文件
  - 机器码/原生码(machine/native code)：CPU可直接执行的代码，可执行二进制程序(executables)

数的表示 - 进制(Number Systems)

- 十进制(decimal)
- 二进制(binary) 0b（整数除2、小数乘2）
- 八进制(octal) 0
- 十六进制(hexadecimal) 0x

举例：20=0b10100=024=0x14

数的表示 - 计算机的算术

- 原码(sign-magnitude)：最高位为符号位
- 反码(**ones'** complement)：原码按位取反
- 补码(**two's** complement)：反码+1

数的表示 - 计算机的算术

- 原码(sign-magnitude)：最高位为符号位
- 反码(**ones'** complement)：原码按位取反
- 补码(**two's** complement)：反码+1

* 浮点数

符号S,1	阶码E,8,移码	尾数F,23,原码

移码偏置常数为127（单精度）、1023（双精度）

$$(-1)^S \times 1.F \times 2^{E-127}$$

计算机编码

- ASCII (American Standard Code for Information Interchange)
  - 7位或8位二进制数
  - ‘0’:48, ‘A’:65, ‘a’:97
- Unicode (统一码/万国码/单一码) / UTF-8：为每种语言中的每个字符设定了统一且唯一的二进制编码，以满足跨语言、跨平台进行文本转换、处理的要求

3.2

变量

变量名

- 长度限定：任意长，但只会处理
  - C89：≤ 31字符的内部标识符，≤ 6字符的外部标识符
  - C99：≤ 63字符的内部标识符，≤ 31字符的外部标识符

变量名

- 长度限定：任意长，但只会处理
  - C89：≤ 31字符的内部标识符，≤ 6字符的外部标识符
  - C99：≤ 63字符的内部标识符，≤ 31字符的外部标识符
- 可用字符：大小写$52+$阿拉伯数字$10+$下划线$1=63$
- 组合规则:
  - 第一个字符不可数字（也即可以是字母或下划线）
  - 但避免_开头，因操作系统和C标准库一般以_开头（不是环境变量）

变量名

- 长度限定：任意长，但只会处理
  - C89：≤ 31字符的内部标识符，≤ 6字符的外部标识符
  - C99：≤ 63字符的内部标识符，≤ 31字符的外部标识符
- 可用字符：大小写$52+$阿拉伯数字$10+$下划线$1=63$
- 组合规则:
  - 第一个字符不可数字（也即可以是字母或下划线）
  - 但避免_开头，因操作系统和C标准库一般以_开头（不是环境变量）
- 关键字：全小写
- 可以有相同的变量名与函数名，不会报错

局部变量与全局变量

- 局部变量：只在作用域内起作用
  - 最近的大括号对内，for循环不写大括号也算
- 全局/外部变量：作用域由定义处开始到程序结束，用extern修饰

局部变量与全局变量

- 局部变量：只在作用域内起作用
  - 最近的大括号对内，for循环不写大括号也算
  - 记得初始化，否则值为不确定的值
- 全局/外部变量：作用域由定义处开始到程序结束，用extern修饰

局部变量与全局变量

- 局部变量：只在作用域内起作用
  - 最近的大括号对内，for循环不写大括号也算
  - 记得初始化，否则值为不确定的值
  - 避免在函数内返回局部变量，主要是指针类型，如数组
- 全局/外部变量：作用域由定义处开始到程序结束，用extern修饰

局部变量与全局变量

- 局部变量：只在作用域内起作用
  - 最近的大括号对内，for循环不写大括号也算
  - 记得初始化，否则值为不确定的值
  - 避免在函数内返回局部变量，主要是指针类型，如数组
- 全局/外部变量：作用域由定义处开始到程序结束，用extern修饰
  - 不跨文件在程序前文声明了，则可不写修饰符

局部变量与全局变量

- 局部变量：只在作用域内起作用
  - 最近的大括号对内，for循环不写大括号也算
  - 记得初始化，否则值为不确定的值
  - 避免在函数内返回局部变量，主要是指针类型，如数组
- 全局/外部变量：作用域由定义处开始到程序结束，用extern修饰
  - 不跨文件在程序前文声明了，则可不写修饰符
  - 全局变量自动初始化为0（数值型变量）、空字符（字符变量）

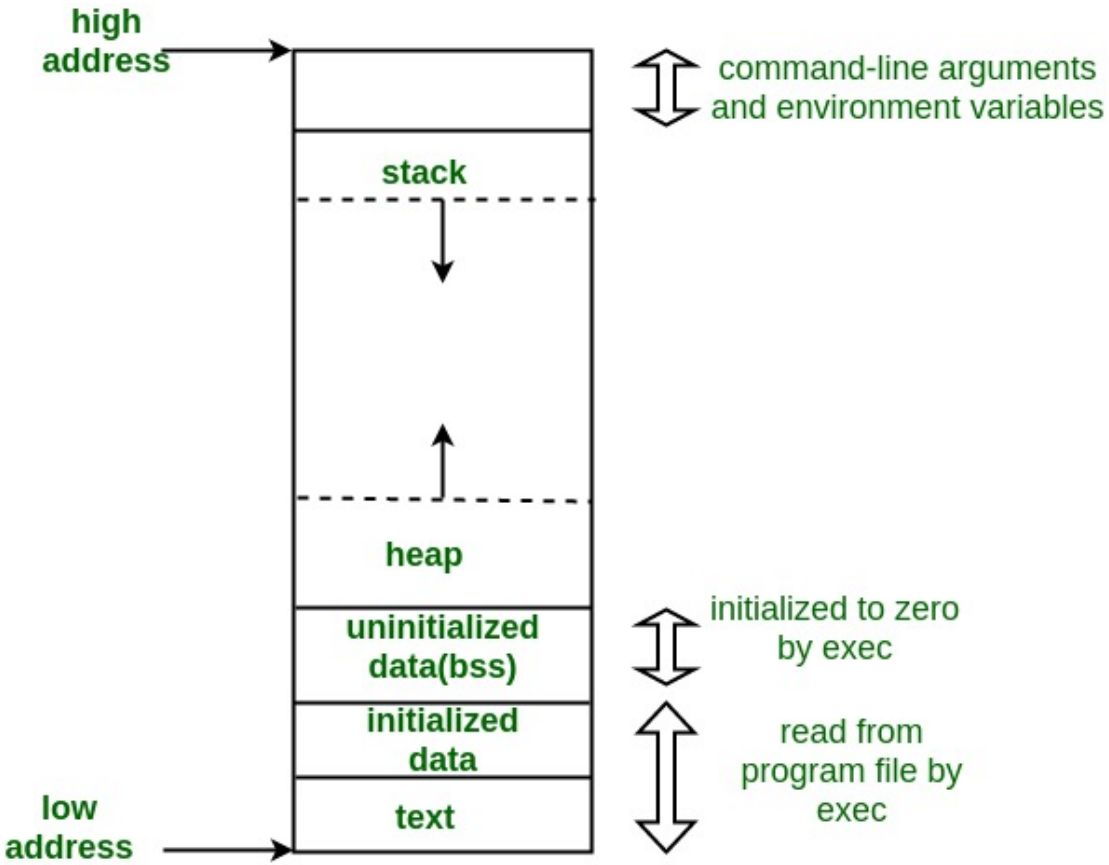
局部变量与全局变量

- 局部变量：只在作用域内起作用
  - 最近的大括号对内，for循环不写大括号也算
  - 记得初始化，否则值为不确定的值
  - 避免在函数内返回局部变量，主要是指针类型，如数组
- 全局/外部变量：作用域由定义处开始到程序结束，用extern修饰
  - 不跨文件在程序前文声明了，则可不写修饰符
  - 全局变量自动初始化为0（数值型变量）、空字符（字符变量）
  - extern后面要不要加类型（如extern int i），看不同版本编译器

静态变量与自动变量

- 静态变量：`static`
  在静态存储区分配存储单元，返回函数调用时值不会被清除，在程序整个运行期间都不释放
  - 变量、函数、结构体全部都可以static
  - 不可`static static int`
- 自动变量：`auto`
  在动态存储区分配存储单元，每次函数调用新建，结束后即释放
  - 没有加修饰词的都是自动变量，`auto`可省略
  - * 区别C++11中的auto关键字

进程的内存分布(Memory Layout)

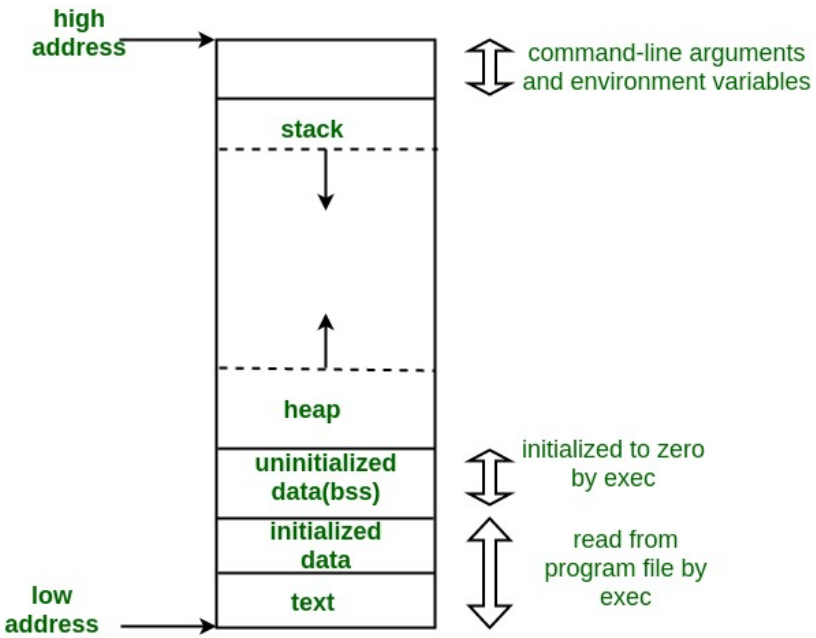


数据和代码都
无区别地用二进制存储

[1] https://www.geeksforgeeks.org/memory-layout-of-c-program/

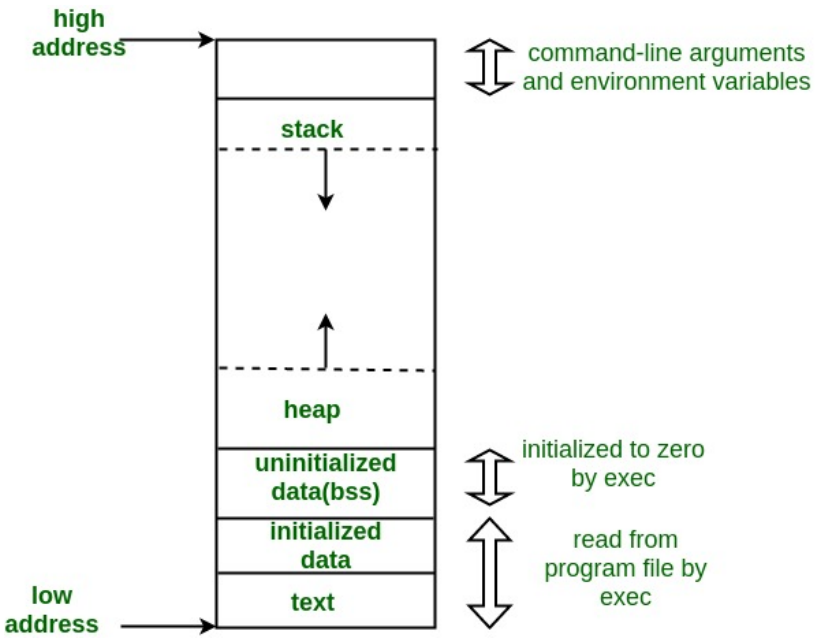
进程的内存分布(Memory Layout)

- 文本/代码(Code)段：包含可执行指令



[1] https://www.geeksforgeeks.org/memory-layout-of-c-program/

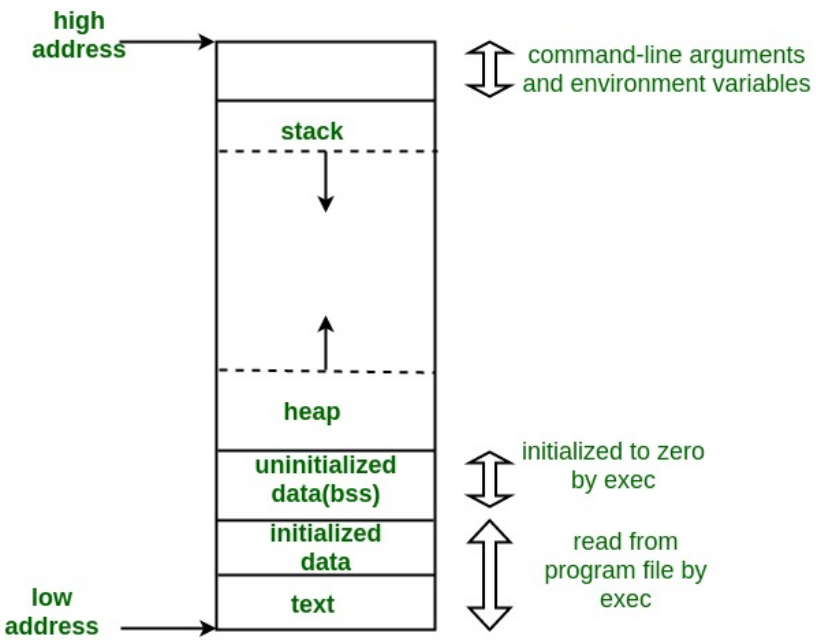
进程的内存分布(Memory Layout)



- 文本/代码(Code)段：包含可执行指令
- 数据(Data)段：包含**已初始化的全局变量、静态变量**，并非只读区间

[1] https://www.geeksforgeeks.org/memory-layout-of-c-program/

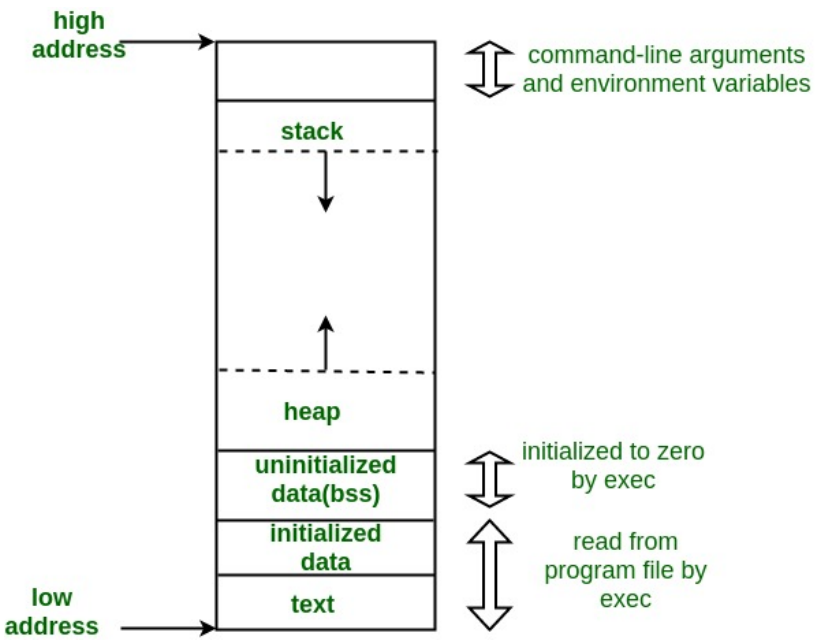
进程的内存分布(Memory Layout)



- 文本/代码(Code)段：包含可执行指令
- 数据(Data)段：包含**已初始化的全局变量、静态变量**，并非只读区间
- BSS(Block Started by Symbol)段：**未初始化的全局变量、静态变量**

[1] https://www.geeksforgeeks.org/memory-layout-of-c-program/

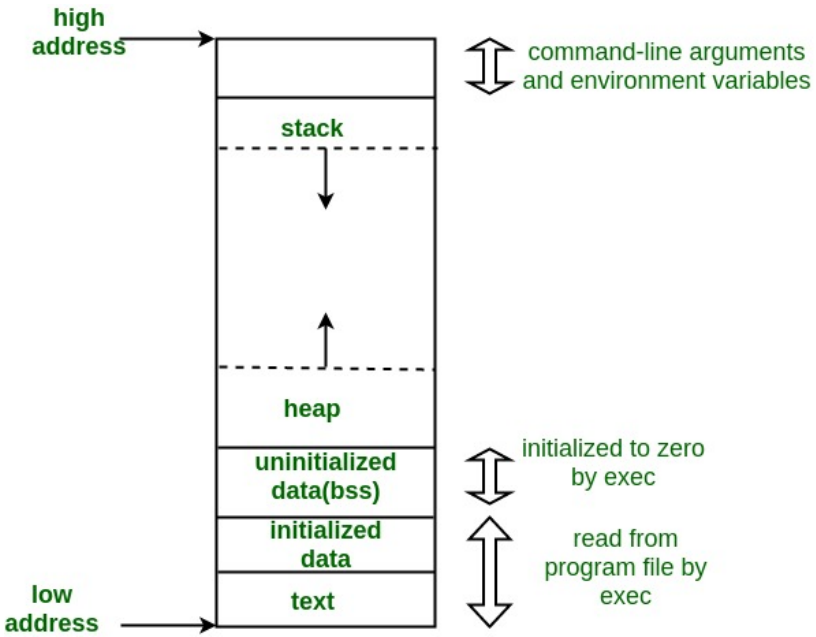
进程的内存分布(Memory Layout)



- 文本/代码(Code)段：包含可执行指令
- 数据(Data)段：包含**已初始化的全局变量、静态变量**，并非只读区间
- BSS(Block Started by Symbol)段：**未初始化的全局变量、静态变量**
- 栈(Stack)段：先进后出，用于函数调用，包含**自动变量**

[1] https://www.geeksforgeeks.org/memory-layout-of-c-program/

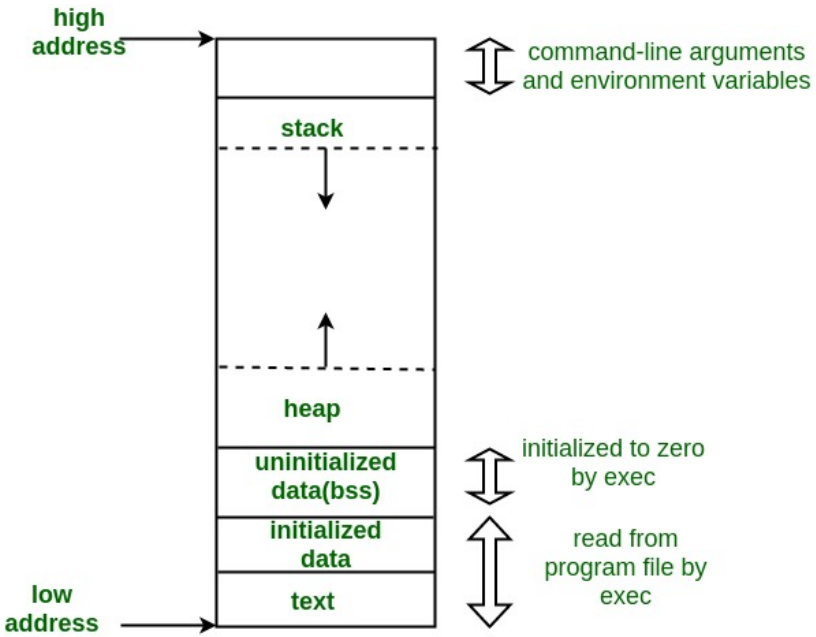
进程的内存分布(Memory Layout)



- 文本/代码(Code)段：包含可执行指令
- 数据(Data)段：包含**已初始化的全局变量、静态变量**，并非只读区间
- BSS(Block Started by Symbol)段：**未初始化的全局变量、静态变量**
- 栈(Stack)段：先进后出，用于函数调用，包含**自动变量**
  - 决定了你能开多大的局部变量、能递归函数多少次

[1] https://www.geeksforgeeks.org/memory-layout-of-c-program/

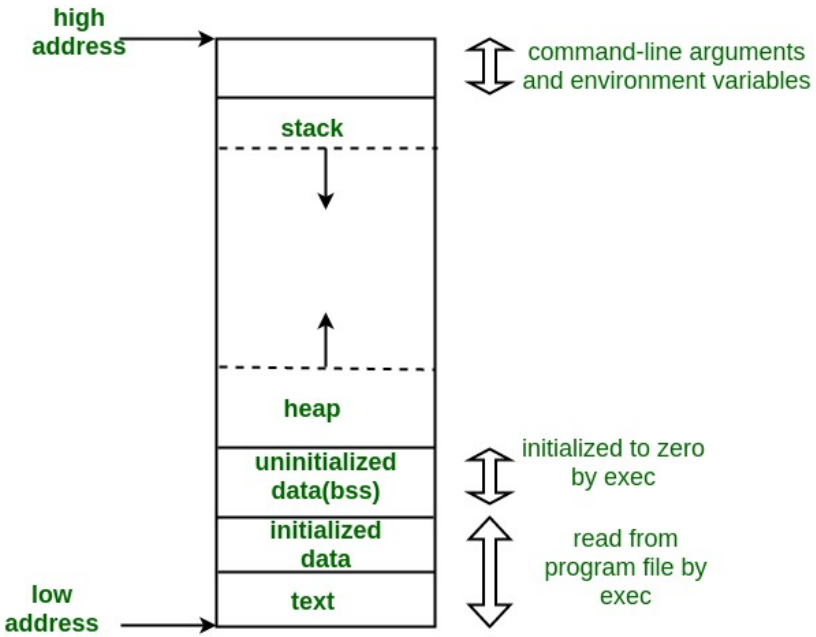
进程的内存分布(Memory Layout)



- 文本/代码(Code)段：包含可执行指令
- 数据(Data)段：包含**已初始化的全局变量、静态变量**，并非只读区间
- BSS(Block Started by Symbol)段：**未初始化的全局变量、静态变量**
- 栈(Stack)段：先进后出，用于函数调用，包含**自动变量**
  - 决定了你能开多大的局部变量、能递归函数多少次
  - 栈段一般是8KB，开大数组一定要开全局!

[1] https://www.geeksforgeeks.org/memory-layout-of-c-program/

进程的内存分布(Memory Layout)



- 文本/代码(Code)段：包含可执行指令
- 数据(Data)段：包含**已初始化的全局变量、静态变量**，并非只读区间
- BSS(Block Started by Symbol)段：**未初始化的全局变量、静态变量**
- 栈(Stack)段：先进后出，用于函数调用，包含**自动变量**
  - 决定了你能开多大的局部变量、能递归函数多少次
  - 栈段一般是8KB，开大数组一定要开全局!
- 堆(Heap)段：动态内存分配，用`malloc`、`realloc`、`free`管理

[1] https://www.geeksforgeeks.org/memory-layout-of-c-program/

定义与声明

```
int i = 5;
extern int i;
```

定义与声明

```
int i = 5;
extern int i;
```

- 定义(definition)：指编译器创建一个对象，为这个对象**分配**一块内存并取名（即变量名）
  - 定义只可有一次
  - C里没有`string str`
  - C99 `for`循环内可定义变量
- 声明(declaration)：告知编译器这个名字已经匹配到一块内存上，并将这个名字和内存链接起来（没有分配内存）
  - 声明可以出现多次

定义与声明总结[1]

	声明	定义	初始化
`int i;`（局部）	Yes	Yes	No
`int i=5;`（局部）	Yes	Yes	Yes (5)
`int i;`（全局）	Yes	No	Yes (0)
`extern int i;`	Yes	No	No

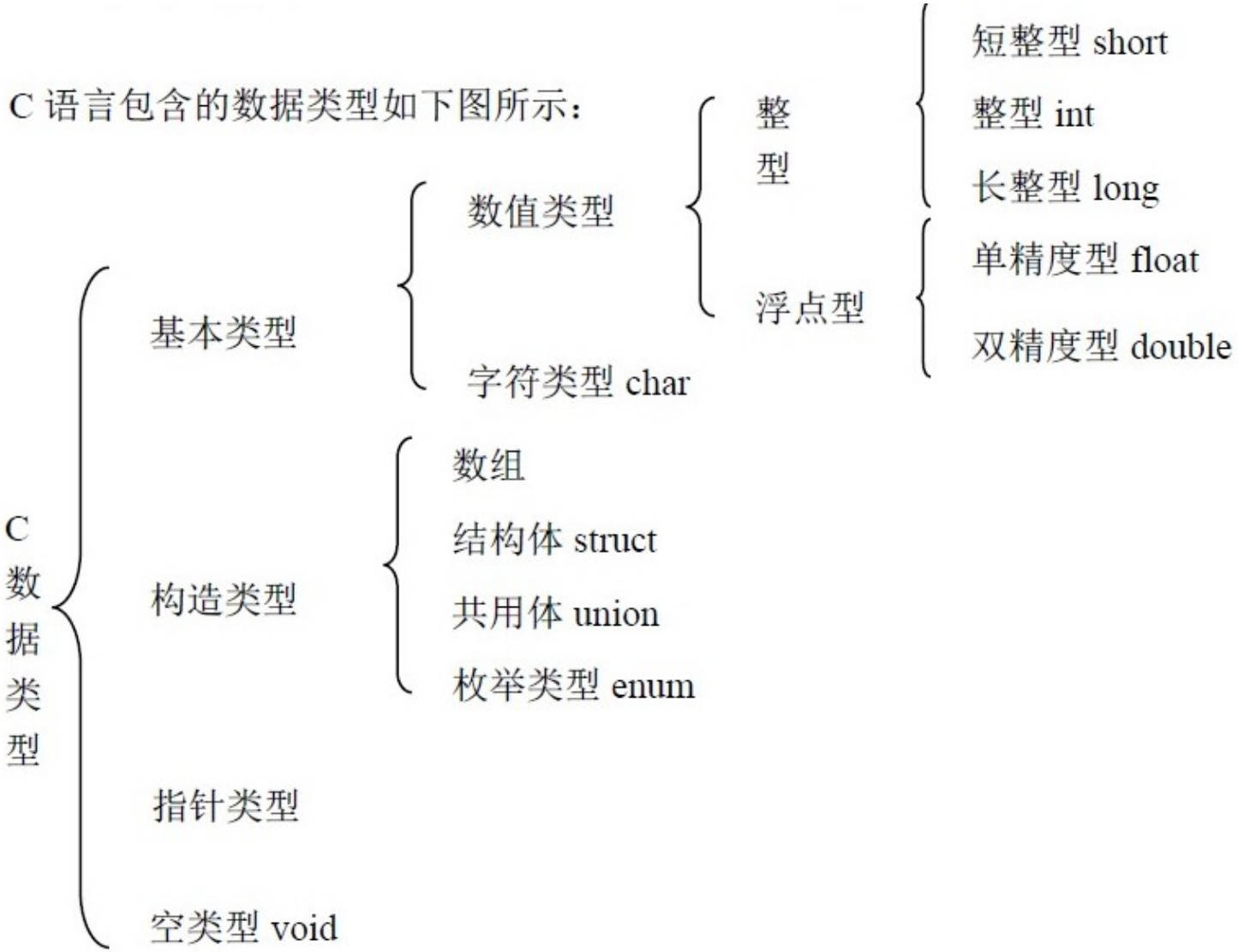
[1] https://stackoverflow.com/questions/4769599/declaration-or-definition-in-c

3.3

数据类型

数据类型

C 语言包含的数据类型如下图所示：



数据类型

- enum：定义类型不占用空间，定义变量才占用
- short int：修饰符(qualifier)+基本数据类型(basic data type)
- float：无声明0.1默认为double，可以强行转换(0.1f)

数据类型

sizeof

- **单目运算符**
- 返回类型的**字节**大小，由系统编译器决定
- 数组sizeof是数组元素sizeof之和，enum不表
- 特别注意区别struct和union
- 指针之后再讲

数据类型

sizeof

- **单目运算符**
- 返回类型的**字节**大小，由系统编译器决定
- 数组sizeof是数组元素sizeof之和，enum不表
- 特别注意区别struct和union
- 指针之后再讲

1字节(Byte)=8位(Bit)，故32位机是4字节

char	1
short	2
int	4
long	4/8
float	4
double	8

数的表示

- `float x` `%d` 出来垃圾值

数的表示

- `float x` %d 出来垃圾值
- `int x` %f 出来为非常小的数，输出即为0

数的表示

- `float x` %d 出来垃圾值
- `int x` %f 出来为非常小的数，输出即为0
- `char a` %d 出来为ASCII码值

数的表示

- `float x` %d 出来垃圾值
- `int x` %f 出来为非常小的数，输出即为0
- `char a` %d 出来为ASCII码值
- `int i=23 < char a=-23` 不一定，看编译器!

数的表示

- `float x` %d 出来垃圾值
- `int x` %f 出来为非常小的数，输出即为0
- `char a` %d 出来为ASCII码值
- `int i=23 < char a=-23` 不一定，看编译器!
- `unsigned int x=-5` `printf("%d",x)` 仍为-5

常量

- 用const进行修饰
- 常量必须初始化，否则编译错误
- \r回车到行首(carriage return)

3.4

算术逻辑表达式

算术 - 除法

- 不同语言对于除法的定义不一样!
- 除法/在C里为**截断**(truncate)取整除法
- **整型**除以**整型**就是**整型**，不会进行自动类型转换

e.g.

$$-9/4 = 9/-4 = -2 \qquad 9/4 = -9/-4 = 2$$

算术 - 自增

- `++a + ++a`一定要有空格，但依然是未定义行为(Undefined Behavior, UB)
- `sizeof(x++)`不会增加x的值[1]

[1] https://stackoverflow.com/questions/8225776/why-does-sizeofx-not-increment-x

算术 - 逻辑

- 逻辑与逻辑或都有**短路**运算，前者为0或1，可判断时将不会执行后续操作

算术 - 逻辑

- 逻辑与逻辑或都有**短路**运算，前者为0或1，可判断时将不会执行后续操作
- 0为假，非零值均为真；但逻辑表达式的返回值只会是0或1

算术 - 逻辑

- 逻辑与逻辑或都有**短路**运算，前者为0或1，可判断时将不会执行后续操作
- 0为假，非零值均为真；但逻辑表达式的返回值只会是0或1
- 按位取反~包括符号位也要取反，故~0=-1区别于!0=1

算术 - 逻辑

- 逻辑与逻辑或都有**短路**运算，前者为0或1，可判断时将不会执行后续操作
- 0为假，非零值均为真；但逻辑表达式的返回值只会是0或1
- 按位取反~包括符号位也要取反，故~0=-1区别于!0=1
- 移位，左移乘2，右移除以2（向下取整）；注意对于有符号数来说是**算术**右移（符号位扩展）

算术 - 其他

- 只有*= /= %= += -= <<= >>= &= |= ^=没有&&=

算术 - 其他

- 只有*= /= %= += -= <<= >>= &= |= ^=没有&&=
- 赋值表达式具有返回值，为赋值成功的值

算术 - 其他

- 只有*= /= %= += -= <<= >>= &= |= ^=没有&&=
- 赋值表达式具有返回值，为赋值成功的值
- 连续赋值是合法的

算术 - 其他

- 只有*= /= %= += -= <<= >>= &= |= ^=没有&&=
- 赋值表达式具有返回值，为赋值成功的值
- 连续赋值是合法的
- 一定看清是几个符号，是==还是=，是&&还是&，并且注意for循环后有无分号

算术 - 其他

- 只有*= /= %= += -= <<= >>= &= |= ^=没有&&=
- 赋值表达式具有返回值，为赋值成功的值
- 连续赋值是合法的
- 一定看清是几个符号，是==还是=，是&&还是&，并且注意for循环后有无分号
- 注意缩进不是C语句的判断标准!

算术 - 其他

- 只有*= /= %= += -= <<= >>= &= |= ^=没有&&=
- 赋值表达式具有返回值，为赋值成功的值
- 连续赋值是合法的
- 一定看清是几个符号，是==还是=，是&&还是&，并且注意for循环后有无分号
- 注意缩进不是C语句的判断标准!
- 逗号表达式取最后一个运算得到的值

算术 - 其他

- 只有*= /= %= += -= <<= >>= &= |= ^=没有&&=
- 赋值表达式具有返回值，为赋值成功的值
- 连续赋值是合法的
- 一定看清是几个符号，是==还是=，是&&还是&，并且注意for循环后有无分号
- 注意缩进不是C语句的判断标准!
- 逗号表达式取最后一个运算得到的值

算术 - 其他

- 只有*= /= %= += -= <<= >>= &= |= ^=没有&&=
- 赋值表达式具有返回值，为赋值成功的值
- 连续赋值是合法的
- 一定看清是几个符号，是==还是=，是&&还是&，并且注意for循环后有无分号
- 注意缩进不是C语句的判断标准!
- 逗号表达式取最后一个运算得到的值

* if比较不会出错的写法是`if(8 == x)`

运算优先级
单算移关与，异或逻条赋

- () [index] -> .
- 单目运算符(右结合)
  ++ -- sizeof & * !(逻辑非)
  ~(按位反) -(负号)
- 算术* / %
- + -
- 移位<< >>
- 关系> < >= <=
- == !=
- 按位与&
- 按位异或^
- 按位或|
- 逻辑与&&
- 逻辑或||
- 条件(三目运算符) ? :
- 赋值+= -= =
- 逗号,

p++相当于(p++)右结合

3.5

控制流

循环

```
for ( init; condition; increment )
{
   statement(s);
}
```

1. init会首先被执行，且只会执行一次

循环

```
for ( init; condition; increment )
{
   statement(s);
}
```

1. init会首先被执行，且只会执行一次
2. 判断condition，如果为真，则执行循环主体；如果为假，则不执行循环主体

循环

```
for ( init; condition; increment )
{
   statement(s);
}
```

1. init会首先被执行，且只会执行一次
2. 判断condition，如果为真，则执行循环主体；如果为假，则不执行循环主体
3. 在执行完for循环主体后，控制流会跳回上面的increment，更新循环控制变量；可以留空，只要在条件后有一个分号出现即可

循环

```
for ( init; condition; increment )
{
   statement(s);
}
```

1. init会首先被执行，且只会执行一次
2. 判断condition，如果为真，则执行循环主体；如果为假，则不执行循环主体
3. 在执行完for循环主体后，控制流会跳回上面的increment，更新循环控制变量；可以留空，只要在条件后有一个分号出现即可
4. 条件再次被判断，不断循环

循环

```
for ( init; condition; increment )
{
   statement(s);
}
```

- 三个位置都可以留空（中间默认为空），但一定要写两个；
- `for(;*a==*b;a++,b++)` 多个循环变量
- `for(short i=1;i>=0;i++)` 到short边界会停
- `for(double k=0.0;k<3.0;k++)` 循环3次

3.6

函数与程序结构

函数

函数调用：保护现场，堆栈执行，恢复现场
注意事项

- 函数名首个字符不可为数字

函数

函数调用：保护现场，堆栈执行，恢复现场
注意事项

- 函数名首个字符不可为数字
- 不能把函数作为参数传入

函数

函数调用：保护现场，堆栈执行，恢复现场
注意事项

- 函数名首个字符不可为数字
- 不能把函数作为参数传入
- 函数**返回类型**可以不写，默认为int

函数

函数调用：保护现场，堆栈执行，恢复现场
注意事项

- 函数名首个字符不可为数字
- 不能把函数作为参数传入
- 函数**返回类型**可以不写，默认为int
- 函数都是外部的

函数

函数调用：保护现场，堆栈执行，恢复现场
注意事项

- 函数名首个字符不可为数字
- 不能把函数作为参数传入
- 函数**返回类型**可以不写，默认为int
- 函数都是外部的
- 不能在函数内定义函数

3.7

指针与数组

指针

C的精髓

实质是内存单元的编号（字节编码）

指针

C的精髓
实质是内存单元的编号（字节编码）

- 寻址空间为$2^{32} = 4GB$

指针

C的精髓
实质是内存单元的编号（字节编码）

- 寻址空间为$2^{32} = 4GB$
- 指针sizeof全相同，由机器字长决定，32位机为4，64位机为8

指针

C的精髓
实质是内存单元的编号（字节编码）

- 寻址空间为$2^{32} = 4GB$
- 指针sizeof全相同，由机器字长决定，32位机为4，64位机为8
- 指针都可以自增或自减，增减的值为数据类型的大小决定

指针

C的精髓
实质是内存单元的编号（字节编码）

- 寻址空间为$2^{32} = 4GB$
- 指针sizeof全相同，由机器字长决定，32位机为4，64位机为8
- 指针都可以自增或自减，增减的值为数据类型的大小决定
- 指针**之间**只可做**减法**运算

指针

C的精髓
实质是内存单元的编号（字节编码）

- 寻址空间为$2^{32} = 4GB$
- 指针`sizeof`全相同，由机器字长决定，32位机为4，64位机为8
- 指针都可以自增或自减，增减的值为数据类型的大小决定
- 指针**之间**只可做**减法**运算
- 不可`int* p=10`（类型不同），但可以赋值为`int* p=0`（相当于NULL）

指针

C的精髓
实质是内存单元的编号（字节编码）

- 寻址空间为$2^{32} = 4GB$
- 指针sizeof全相同，由机器字长决定，32位机为4，64位机为8
- 指针都可以自增或自减，增减的值为数据类型的大小决定
- 指针**之间**只可做**减法**运算
- 不可`int* p=10`（类型不同），但可以赋值为`int* p=0`（相当于NULL）
- `int i=10;(&i)++;`不可，因`&i`为常量，得将其赋值为变量`int* p=&i`才可以进行自增

指针

C的精髓
实质是内存单元的编号（字节编码）

- 寻址空间为$2^{32} = 4GB$
- 指针sizeof全相同，由机器字长决定，32位机为4，64位机为8
- 指针都可以自增或自减，增减的值为数据类型的大小决定
- 指针**之间**只可做**减法**运算
- 不可`int* p=10`（类型不同），但可以赋值为`int* p=0`（相当于NULL）
- `int i=10;(&i)++;`不可，因`&i`为常量，得将其赋值为变量`int* p=&i`才可以进行自增
- `int* a,b`只有a为指针

指针

C的精髓
实质是内存单元的编号（字节编码）

- 寻址空间为$2^{32} = 4GB$
- 指针sizeof全相同，由机器字长决定，32位机为4，64位机为8
- 指针都可以自增或自减，增减的值为数据类型的大小决定
- 指针**之间**只可做**减法**运算
- 不可`int* p=10`（类型不同），但可以赋值为`int* p=0`（相当于NULL）
- `int i=10;(&i)++;`不可，因`&i`为常量，得将其赋值为变量`int* p=&i`才可以进行自增
- `int* a,b`只有a为指针
- 函数内要改变变量值一定要以指针形式*a=b，而a=&b就不可

指针的结合[2]

[]优先级高于*

- `int **a[10]` → `int **[10] a`
- `int (**a)[10]` → `int [10] **a`
- `int *(*a)[10]` → `int *[10] *a`

[2]This slide is borrowed from Wu Kan.

指针的结合[2]

[]优先级高于*

- `int **a[10]` → `int **[10] a`
- `int (**a)[10]` → `int [10] **a`
- `int *(*a)[10]` → `int *[10] *a`

结合const，遇到变量转为is a，遇到*转为pointer to

- 常量指针：指向常量的指针，`const int* p`与`int const* p`相同
- 指针常量：指针就是常量，`int* const p`

[2]This slide is borrowed from Wu Kan.

函数指针

函数指针可以避免直接呼叫函数名

<stdlib.h>中定义

```
void qsort(void* base, size_t num, size_t width,
int(__cdecl* compare)(const void*,const void*));

int cmp1(const void * a,const void * b)
{
    return (*(int*)a-*(int*)b);
}
qsort(a,10,sizeof(int),&cmp1);
```

数组

- a[i]等价于i[a]

数组

- a[i]等价于i[a]
- int arr[5] = {5};只会将第一位赋值为5
  int arr[5] = {0};则全部赋值为0

数组

- a[i]等价于i[a]
- int arr[5] = {5};只会将第一位赋值为5
  int arr[5] = {0};则全部赋值为0
- int arr[4] = {1, 2, 3, 4}; int p[4]; p = arr;是不允许的

数组

- a[i]等价于i[a]
- int arr[5] = {5};只会将第一位赋值为5
  int arr[5] = {0};则全部赋值为0
- int arr[4] = {1, 2, 3, 4}; int p[4]; p = arr;是不允许的
- 不可创建void数组

数组

- a[i]等价于i[a]
- int arr[5] = {5};只会将第一位赋值为5
  int arr[5] = {0};则全部赋值为0
- int arr[4] = {1, 2, 3, 4}; int p[4]; p = arr;是不允许的
- 不可创建void数组
- 不可int a[2][3] = {1, 2, 3, , 4, 5};

数组

- a[i]等价于i[a]
- int arr[5] = {5};只会将第一位赋值为5
  int arr[5] = {0};则全部赋值为0
- int arr[4] = {1, 2, 3, 4}; int p[4]; p = arr;是不允许的
- 不可创建void数组
- 不可int a[2][3] = {1, 2, 3, , 4, 5};
- 自定义负数索引

```
int arr[4] = {1, 2, 3, 4};
int *p = arr + 3;
printf("%d\n", p[-2]);
```

数组

- a[i]等价于i[a]
- int arr[5] = {5};只会将第一位赋值为5
  int arr[5] = {0};则全部赋值为0
- int arr[4] = {1, 2, 3, 4}; int p[4]; p = arr;是不允许的
- 不可创建void数组
- 不可int a[2][3] = {1, 2, 3, , 4, 5};
- 自定义负数索引

```
int arr[4] = {1, 2, 3, 4};
int *p = arr + 3;
printf("%d\n", p[-2]);
```

- 字符数组逐个读入时记得加\0

二维数组

`a[i][j]` $\rightarrow$ `*(ki+j)`，第二维是分配内存的长度，第一维是分配内存的倍数

- `void f(int n, int a[][10])`
- `void f(int n, int (*a) [10])`
- `void f(int n, int **a)`

3.8

其他

程序异常

- 编译错误(CE)：语法错（括号不匹配、漏;等）
- 运行错误(RE)：数组越界、除数为0、堆栈溢出、访问无效内存等

printf

- `printf("%3d",num);` 补齐位宽
- `printf("%03d",num);` 自动补零

4

Linux常用指令

Linux操作指令

- 编译 `gcc hello.c -o hello`
- 执行 `./hello`
- 返回上级目录 `cd ..`
- 跳转下级目录 `cd Desktop`
- 查看目录内容 `ls`

5

在线测试

在线测试

- 不允许使用C编译器，理论考试也不提供
- 当然，用python是可以的(