- C
  - 主题
  - 主题
  - 原码，反码，补码
    - 正整数的原码、反码、补码bai完全一样，即符号位固定为0，数值位相同。
    - 负整数的符号位固定为1，由原码变为补码时，规则如下
      - 原码符号位1不变，整数的每一位二进制数位求反，得到反码。
      - 反码符号位1不变，反码数值位最低位加1，得到补码。
  - 进制
    - 进制类型
      - 0开头的数组，比如077，八进制
      - 0x开头的数字，比如0x77，十六进制
      - b开头，二进制
    - 进制转换
      - 其他进制转为十进制
        - 该位数字* 该进制的数字 ^ 该位所在的下标
      - 十进制转其他进制
        - 转哪个进制就对该数字除以哪个数字并取余，比如除二取余
      - 二进制转为其他进制
        - 转8进制：取三合一法，以小数点为分界线，向左向右取三位
        - 转16进制：取四合一法，以小数点为分界线，向左向右取四位
        - 例：即(11010111.0100111)B=(327.234)O。
  - 关键字
    - auto
    - extern
    - static
    - typedef
  - 占位符
    - %d，%f,%s,%c,%p...
  - 转义符
    - \n，换行,\'，单引号,\?，问号,\0oo，八进制
  - 运算符
    - 算数运算符法
    - 关系运算符
    - 特殊运算符
      - 逗号，逗号运算会将它两边的表达式按照从左到右的顺序执行。
      - 三元运算符：ary = a != 1? 2: 3
  - 类型
    - 类型转换
      - 从较小类型转换为较大类型，所以这些转换被称为升级，比如unsined转为int
      - 类型升级通常都不会有什么问题，但是类型降级会导致真正的麻烦。原因很简单：较低类型可能放不下整个数字
    - 强制类型转换
    - 数据类型
      - 数字类型：int，float，long，short，unsiged,double，它们之间可以组合使用来表示一种类型
      - 字符型
        - int8_t,int16_t...只是int的别名
        - char，跟编译器有关系，标准占1字节
      - 布尔类型
        - 1.通过#include "stdbool.h"库来声明
        - 2、通过typedef
        - 3、通过宏定义
      - auto
        - 自动判断变量类型
      - typeof
        - 它可以取得变量的类型，或者表达式的类型。通常用来声明不确定类型的变量
  - 字符串
    - 数组型字符串
      - 初始化字符串时，要给字符串变量的元素个数比字符串长度大1（为\0预留）:char words[12] = "i am a good"; //只有11个字符，但是要至少预留12个
      - char ar1[] = "i am a hello world";
      - 字符串数组的名称也是该数组首元素的地址
    - 指针型字符串
      - const char *pt1 = "i am a hello world";
      - 指针类型的字符串效率更高，但是无法更改
  - 枚举
    - 声明符号名称来表示整型常量。enum 常量是int类型，它的目的是提高程序的可读性
    - enum spectrum {red, orange, yellow, green, blue, violet}; // 默认值为0,1,2,3,4,5
    - typedef enum spec{}
  - 数组
    - 如果数组只声明未初始化，那么编译器会自动将它们初始化未0
    - 不允许数组之间相互赋值，即使完全相同。也不允许一次性赋多个值
    - 初始化器
      - int arr[6] = {[5] = 212}; // 只把arr[5]初始化为212
    - 数组指针
      - 数组名是数组首元素的地址。所以下面的语句成立：flizny == &flizny[0];
  - 作用域
  - 流程控制
    - while
      - do while
    - for
    - switch
      - 匹配到的case如果没有break，则会按顺序执行下一个case。否则会跳出循环
      - 可以匹配多个case，多个case之间按顺序执行
      - 所有case都匹配不到，执行default
    - goto，不建议使用
    - continue
    - break
  - 结构体
    - 声明：struck book {}
    - 使用：struct book book_obj;
    - 结构体数组
      - struct book book_objs[40];
    - 嵌套结构体
    - 指向结构体的指针
      - struct book *book_p
        - 通过指针访问结构体数据时，使用->。比如：book_p->name
    - 结构体当作参数
      - 传递结构体
      - 传递结构体指针
    - 结构体数组
      - 声明：struct funds {};，使用：struct funds jones[2] = {
        {"Garlic-Melon Bank", 4032.27, "Lucky's Savings and Loan",8543.94},
        {"Honest Jack's Bank", 3620.88, "Party Time Savings", 3802.91 }};
    - 匿名结构体
      - 声明：struct person {
        int id;
        struct {char first[20]; char last[20];}; // 匿名结构
        };
      - 使用：// 访问匿名结构体的属性
        puts(ted.first);
    - 其他特性
      - 结构体既可以当作参数传入，也可以当作返回值返回
      - 同一结构体声明的多个变量之间可以互相赋值
  - typedef
    - typedef char * STRING;
    - 自定义结构体的名字以最后结尾为准：typedef struct tagPOINT {}POINT; 使用的话：POINT p1
  - 指针
    - &，取地址符
    - *，解引用
    - 指针使用前必须要先初始化：将它指向一个地址
    - 指针做加减法：根据指针的类型移动对应的地址
    - 指针的表达式从右往左进行：(short*)&f就是先取f的地址
  - 函数
    - 分配的内存在堆中
      - malloc
        - 分配内存
  - 联合
    - 能在同一个内存空间中储存不同的数据类型（但不是同时储存）
    - 子主题
      - 联合数组
  - 内联函数
    - 通常，函数调用都有一定的开销。通过使用宏使代码内联，可以避免这样的开销。内联函数的编译代码会与其他程序代码内联起来，编译器也不需要调到某个函数执行后再跳回来。但是也可能不起作用
    - 内联函数应该比较简洁
  - 类型限定符
    - const
      - 被const修饰的变量，在程序整个生命周期无法被修改
        - 本身不能修改，但是它指向的值可以被修改：float * const pt;
        - 本身跟它指向的值都不能被修改：const float * const ptr;
        - 指向的值不能修改，而 pt 本身的值可以改变。const float * pf
    - Atomic
      - 原子类型必须要通过各种宏函数来访问。当一个线程对一个原子类型的对象执行原子操作时，其他线程不能访问该对象
        - _Atomic int hogs; // hogs 是一个原子类型的变量
          atomic_store(&hogs, 12); //stdatomic.h中的宏。这里，在hogs中储存12是一个原子过程，其他线程不能访问hogs。
    - restrict
      - 只能用于指针，允许编译器优化某部分代码以更好地支持计算
  - 变长数组
    - 可以使用const变量
    - 不能在声明时同时初始化它们，需要使用循环语句来对它进行初始化
    - 举例：int getInt(int n) {
      char ss[n];
      return 0;
      }
      - 在c99之前，这里的n不允许是变量
  - IO流
    - 行缓冲
    - 完全缓冲
    - open，打开一个文件描述符
      - read
      - write
      - lseek
      - stat、lstat获取文件属性
      - opendir，打开目录
  - 网络编程
    - socket
  - gcc
    - 见 g++部分
  - 编译
    - 编译
      - 编译器把源代码转换成中间代码
    - 链接
      - 链接器把中间代码和其他代码合并，生成可执行文件
      - 链接器还将编写的程序和预编译的库代码合并。
    - 整个流程：源代码->编译器->目标代码(.o文件)->链接器(库代码，启动代码)->可执行文件
  - 宏
    - 由#define关键字定义的变量，叫宏
    - 宏在预处理阶段只会是表达式形式，而不会进行真正的运算
  - 预处理
    - #include
      - 将包含的文件在编译阶段导入代码中
        - <>：告诉预处理器在标准系统目录中查找依赖文件
        - ""：告诉预处理器首先在当前目录中（或文件名中指定的其他目录）查找该文件，如果未找到再查找标准系统目录
    - #if、#ifdef、 #ifndef、#else、#elif和#endif
    - 预处理器
      - 预处理之前，编译器必须对该程序进行一些翻译处理
        - 删除注释
        - 删除换行符后面的实例
        - 这些都完成后，才进入预处理阶段
    - 预处理阶段
      - 预处理器查找所有以#号开始的预处理指令
  - 多进程
    - 进程在内存中有三部分组成：数据段、堆栈段和代码段
      - 代码段：就是存放程序代码的数据，如果有数个进程运行同一个一个程序，那么它们就可以使用同一个代码段（代码段是可以共享的）
      - 堆栈段：存放的是子程序的返回地址、参数以及程序的局部变量，主要是保存进程的执行的环境，这里用到了栈先进后出的特性，可以看做具有记忆上一次执行的环境
      - 数据段：存放程序的全局变量 、常数
    - 创建
      - fork()
        - 父进程调用wait、waitpid函数阻塞自己，等待子进程退出
    - 通信
      - 管道
      - 共享内存
        - shmget，创建共享内存
        - attach，连接到共享内存区域
      - 信号量
        - 信号量一般配合共享内存使用，避免多个进程使用同一内存空间时，出现写冲突
      - 消息队列
        - msgget，创建消息队列
        - msgsnd，发送消息
        - msgrcv，接收消息
      - 本地套接字
      - 等
  - 多线程
    - 同一进程中的多条线程将共享该进程中的全部系统资源，如虚拟地址空间，文件描述符和信号处理等等。但同一进程中的多个线程有各自的调用栈，自己的寄存器环境，自己的线程本地存储
      - 子主题
    - 创建
      - pthread_create
        - pthread_join线程等待
  - Makefile核心
    - target
      - makefile中第一次出现的目标，叫做makefile的终极目标。makefile就是为终极目标服务的
    - prerequisites
    - command
      - 必须以tab键开始
  - gdb
    - 启用：gcc编译的命令最后加-g
    - gdb +二进制文件+core