哈希表（Hash table，也叫散列表），是根据关键码值(Key value)而直接进行访问的数据结构。也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。哈希表在像Java、C#等语言中是与生俱来的。可是在C的世界中，似乎只有自己动手，丰衣足食了。

[哈希表](http://www.nowamagic.net/academy/tag/%E5%93%88%E5%B8%8C%E8%A1%A8)实现中需要注意的问题。

**1. 哈希函数**

也叫散列函数，即：根据key，计算出key对应记录的储存位置：position = f(key)

散列函数满足以下的条件：

1. 对输入值运算，得到一个固定长度的摘要(Hash value)；
2. 不同的输入值可能对应同样的输出值；

以下的函数都可以认为是一个散列函数：

* f(x) = x mod 16;
* f(x) = (x2 + 10) \* x;
* f(x) = (x | 0×0000FFFF) XOR (x >> 16);

不过，仅仅满足上面这两条的函数，作为散列函数，还有不足的地方。我们还希望散列函数满足下面几点：

1. 散列函数的输出值尽量接近均匀分布；
2. x的微小变化可以使f(x)发生非常大的变化，即所谓“雪崩效应”([Avalanche effect](http://en.wikipedia.org/wiki/Avalanche_effect))；

上面两点用数学语言表示，就是：

1. 输出值y的分布函数F(y)=y/m, m为散列函数的最大值。或记为y~U[0, m]
2. |df(x)/dx| >> 1；

* 从上面两点，大家看看，前面举例的三个散列函数，哪个更好呢？对了，是第三个：f(x) = (x | 0×0000FFFF) XOR (x >> 16);

它很完美地满足“好的散列函数”的两个附加条件。

**2、哈希冲突（Hash collision）**

也就是两个不同输入产生了相同输出值的情况。首先，哈希冲突是无法避免的，因此，哈希算法的选择直接决定了哈希冲突发送的概率；同时必须要对哈希冲突进行处理，方法主要有以下几种：

1. [链地址法](http://www.nowamagic.net/academy/tag/%E9%93%BE%E5%9C%B0%E5%9D%80%E6%B3%95)。即对Hash表中每个Hash值建立一个冲突表，即将冲突的几个记录以表的形式存储在其中。具体可以参照 [散列冲突处理：链地址法](http://www.nowamagic.net/academy/detail/3008060) 。
2. 开放地址法。具体可以参照 [散列冲突处理：开放定址法](http://www.nowamagic.net/academy/detail/3008050) 。

[view source](http://www.nowamagic.net/academy/detail/3008085#viewSource)

[print](http://www.nowamagic.net/academy/detail/3008085#printSource)[?](http://www.nowamagic.net/academy/detail/3008085#about)

|  |  |  |
| --- | --- | --- |
| 001 | //#include "stdafx.h" | |
| 002 | #include "string.h" |

|  |  |
| --- | --- |
| 003 | #include "stdio.h" |
| 004 | #include "stdlib.h" | |

|  |  |
| --- | --- |
| 005 |  |
| 006 | typedef struct \_node | |

|  |  |
| --- | --- |
| 007 | { |
| 008 | char \*name; | |

|  |  |
| --- | --- |
| 009 | char \*desc; |
| 010 | struct \_node \*next; | |

|  |  |  |
| --- | --- | --- |
| 011 | } node; | |
| 012 |  |

|  |  |
| --- | --- |
| 013 | #define HASHSIZE 101 |
| 014 | static node\* hashtab[HASHSIZE]; | |

|  |  |
| --- | --- |
| 015 |  |
| 016 | void inithashtab() | |

|  |  |
| --- | --- |
| 017 | { |
| 018 | int i; | |

|  |  |  |
| --- | --- | --- |
| 019 | for(i=0; i < HASHSIZE; i++) | |
| 020 | hashtab[i]=NULL; |

|  |  |
| --- | --- |
| 021 | } |
| 022 |  | |

|  |  |  |
| --- | --- | --- |
| 023 | unsigned int hash(char \*s) | |
| 024 | { |

|  |  |  |
| --- | --- | --- |
| 025 | unsigned int h=0; | |
| 026 | for(; \*s; s++) |

|  |  |
| --- | --- |
| 027 | h=\*s+h\*31; |
| 028 | return h%HASHSIZE; | |

|  |  |
| --- | --- |
| 029 | } |
| 030 |  | |

|  |  |  |
| --- | --- | --- |
| 031 | node\* lookup(char \*n) | |
| 032 | { |

|  |  |  |
| --- | --- | --- |
| 033 | unsigned int hi=hash(n); | |
| 034 | node\* np=hashtab[hi]; |

|  |  |  |
| --- | --- | --- |
| 035 | for(; np!=NULL; np=np->next) | |
| 036 | { |

|  |  |  |
| --- | --- | --- |
| 037 | if(!strcmp(np->name,n)) | |
| 038 | return np; |

|  |  |  |
| --- | --- | --- |
| 039 | } | |
| 040 |  |

|  |  |  |
| --- | --- | --- |
| 041 | return NULL; | |
| 042 | } |

|  |  |
| --- | --- |
| 043 |  |
| 044 | char\* m\_strdup(char \*o) | |

|  |  |
| --- | --- |
| 045 | { |
| 046 | int l=strlen(o)+1; | |

|  |  |  |
| --- | --- | --- |
| 047 | char \*ns=(char\*)malloc(l\*sizeof(char)); | |
| 048 | strcpy(ns,o); |

|  |  |
| --- | --- |
| 049 | if(ns==NULL) |
| 050 | return NULL; | |

|  |  |
| --- | --- |
| 051 | else |
| 052 | return ns; | |

|  |  |
| --- | --- |
| 053 | } |
| 054 |  | |

|  |  |  |
| --- | --- | --- |
| 055 | char\* get(char\* name) | |
| 056 | { |

|  |  |  |
| --- | --- | --- |
| 057 | node\* n=lookup(name); | |
| 058 | if(n==NULL) |

|  |  |  |
| --- | --- | --- |
| 059 | return NULL; | |
| 060 | else |

|  |  |  |
| --- | --- | --- |
| 061 | return n->desc; | |
| 062 | } |

|  |  |
| --- | --- |
| 063 |  |
| 064 | int install(char\* name,char\* desc) | |

|  |  |
| --- | --- |
| 065 | { |
| 066 | unsigned int hi; | |

|  |  |
| --- | --- |
| 067 | node\* np; |
| 068 | if((np=lookup(name))==NULL) | |

|  |  |
| --- | --- |
| 069 | { |
| 070 | hi=hash(name); | |

|  |  |  |
| --- | --- | --- |
| 071 | np=(node\*)malloc(sizeof(node)); | |
| 072 | if(np==NULL) |

|  |  |
| --- | --- |
| 073 | return 0; |
| 074 | np->name=m\_strdup(name); | |

|  |  |  |
| --- | --- | --- |
| 075 | if(np->name==NULL) return 0; | |
| 076 | np->next=hashtab[hi]; |

|  |  |  |
| --- | --- | --- |
| 077 | hashtab[hi]=np; | |
| 078 | } |

|  |  |
| --- | --- |
| 079 | else |
| 080 | free(np->desc); | |

|  |  |
| --- | --- |
| 081 | np->desc=m\_strdup(desc); |
| 082 | if(np->desc==NULL) return 0; | |

|  |  |
| --- | --- |
| 083 |  |
| 084 | return 1; | |

|  |  |
| --- | --- |
| 085 | } |
| 086 |  | |

|  |  |
| --- | --- |
| 087 | /\* A pretty useless but good debugging function, |
| 088 | which simply displays the hashtable in (key.value) pairs | |

|  |  |
| --- | --- |
| 089 | \*/ |
| 090 | void displaytable() | |

|  |  |
| --- | --- |
| 091 | { |
| 092 | int i; | |

|  |  |
| --- | --- |
| 093 | node \*t; |
| 094 | for(i=0; i < HASHSIZE; i++) | |

|  |  |
| --- | --- |
| 095 | { |
| 096 | if(hashtab[i]==NULL) | |

|  |  |  |
| --- | --- | --- |
| 097 | printf("()"); | |
| 098 | else |

|  |  |
| --- | --- |
| 099 | { |
| 100 | t=hashtab[i]; | |

|  |  |
| --- | --- |
| 101 | printf("("); |
| 102 | for(; t!=NULL; t=t->next) | |

|  |  |  |
| --- | --- | --- |
| 103 | printf("(%s.%s) ",t->name,t->desc); | |
| 104 | printf(".)"); |

|  |  |  |
| --- | --- | --- |
| 105 | } | |
| 106 | } |

|  |  |
| --- | --- |
| 107 | } |
| 108 |  | |

|  |  |  |
| --- | --- | --- |
| 109 | void cleanup() | |
| 110 | { |

|  |  |
| --- | --- |
| 111 | int i; |
| 112 | node \*np,\*t; | |

|  |  |  |
| --- | --- | --- |
| 113 | for(i=0; i < HASHSIZE; i++) | |
| 114 | { |

|  |  |  |
| --- | --- | --- |
| 115 | if(hashtab[i]!=NULL) | |
| 116 | { |

|  |  |
| --- | --- |
| 117 | np=hashtab[i]; |
| 118 | while(np!=NULL) | |

|  |  |
| --- | --- |
| 119 | { |
| 120 | t=np->next; | |

|  |  |
| --- | --- |
| 121 | free(np->name); |
| 122 | free(np->desc); |

|  |  |  |
| --- | --- | --- |
| 123 | free(np); | |
| 124 | np=t; |

|  |  |  |
| --- | --- | --- |
| 125 | } | |
| 126 | } |

|  |  |  |
| --- | --- | --- |
| 127 | } | |
| 128 | } |

|  |  |
| --- | --- |
| 129 |  |
| 130 | main() | |

|  |  |
| --- | --- |
| 131 | { |
| 132 | int i; | |

|  |  |
| --- | --- |
| 133 | char\* names[]= {"name","address","phone","k101","k110"}; |
| 134 | char\* descs[]= {"Sourav","Sinagor","26300788","Value1","Value2"}; | |

|  |  |
| --- | --- |
| 135 |  |
| 136 | inithashtab(); | |

|  |  |
| --- | --- |
| 137 | for(i=0; i < 5; i++) |
| 138 | install(names[i],descs[i]); | |

|  |  |
| --- | --- |
| 139 |  |
| 140 | printf("Done"); | |

|  |  |  |
| --- | --- | --- |
| 141 | printf("If we didnt do anything wrong..""we should see %s\n",get("k110")); | |
| 142 |  |

|  |  |  |
| --- | --- | --- |
| 143 | install("phone","9433120451"); | |
| 144 |  |

|  |  |  |
| --- | --- | --- |
| 145 | printf("Again if we go right, we have %s and %s",get("k101"),get("phone")); | |
| 146 |  |

|  |  |  |
| --- | --- | --- |
| 147 | /\*displaytable();\*/ | |
| 148 | cleanup(); |

|  |  |  |
| --- | --- | --- |
| 149 | return 0; | |
| 150 | } |