

School of Computer Science and Software Engineering  
Clayton Campus, Monash University

**CSE1303 Part A**  
**Summer Semester, 2001**

## Practical Session 6: Hash Tables



### ***Aim of this Practical Session***

- To familiarise yourself with Hash Tables and how to implement them.

**Project:** To write a simple spell checker.

### ***Background:***

In this prac will use a Hash Table to implement a simple spell checker. The idea behind the program is to read a dictionary into a Hash Table, then check every word in a nominated file to see whether it is in the Hash Table. Any word not found will be assumed to be misspelt and will be printed.

### ***Preparation (3 marks if completed before class)***

Before this prac:

1. Using the following function,  
`unsigned hash(char* s)`

```
{
    int i = 0;
    unsigned value = 0;

    while (s[i] != '\0')
    {
        value = (value + 7*s[i]) % 11;
        i++;
    }
    return value;
}
```

calculate the hash values of the following names:

**Eva, Amy, Tim, Ron, Jan, Kim, Dot, Ann, Jim, Jon**

2. Why are unsigned variables being used in the previous hash function?
3. Write a C program which uses a Hash Table, of size **11**, and the above hash function, to store character strings.
  - **You may assume that character strings have at most 30 characters.**

**Question 1: (3 marks)**

Write C program which uses a Hash Table of size, **24593**, the following hash function:

```
#define TABLESIZE 24593
#define PRIME 12289

unsigned hash(char* s)
{
    int i = 0;
    unsigned value = 0;

    while (s[i] != '\0')
    {
        value = (value + PRIME*s[i]) % TABLESIZE;
        i++;
    }
    return value;
}
```

and has a menu with the following options:

- **Read** which asks the user for a filename, opens the file, and stores all the character strings in the file in the Hash Table **Search**: which asks the user for a character string, and then prints a message indicating whether or not the string is contained in the Hash Table.
- **List**: which prints out all the character strings stored in the Hash Table.
- **Quit**: which allows the user to quit the menu and exit the program.

➤ *Make sure you are dealing with collisions when you are inserting and searching – use linear probing, or else the advanced question is pointless!*

**Question 2: (4 marks)**

Change the program you wrote in Question 1, so that instead of a menu the program does the following steps:

1. Read in all the words stored in the file **dict.txt**, which can found at <http://www.csse.monash.edu.au/courseware/cse1303s/parta/prac/prac06/>, and store these words in the Hash Table.
2. Ask the user for a filename and open the file.
3. Read in each line of the file. For any line which contains a misspelt word (i.e. one not in the Hash Table), print out the line number together with all the words misspelt on that line.  
(you will need to use `fgets` and `sscanf` for this question)

**Advanced Questions: (2 marks)**

In the above implementation of a Hash Table the size of the table was fixed. In this part of the prac we extend the implementation so that the size of the table is dynamic.

**Questions:**

- In your implementation of a Hash Table add the operation **resize**, which changes the size of the Hash Table. This operation will require a parameter **newSize**, which will be the size of the new Hash Table.
  - ❖ **When you change the size of the Hash Table you will need to insert all the items from the old Hash Table into the new Hash Table.**
  - ❖ **You will need to change the hash function when you change the size of the Hash Table. This means TABLESIZE will become a variable. For this part you may assume that only the TABLESIZE changes, and that the parameter PRIME remains constant.**
  - ❖ **Make sure you deallocate any memory you no longer require.**
- Change the program you wrote in Question 2, so that **PRIME** is now a variable, the initial size of the Hash Table is **11**, and the **PRIME** is initially set to **7**. Then, each time the table gets full, increase the size of table and the **PRIME** variable to be the next prime in the following list:

**7, 11, 23, 53, 97, 193, 389, 769, 1543, 3079, 6151, 12289, 245**