Symbol Tables

anhtt-fit@mail.hut.edu.vn

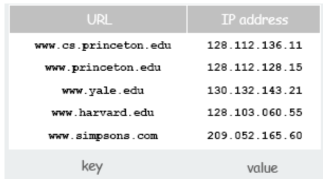
ADT

Key-value pair abstraction.

- Insert a value with specified key.
- Given a key, search for the corresponding value.

Example: DNS lookup.

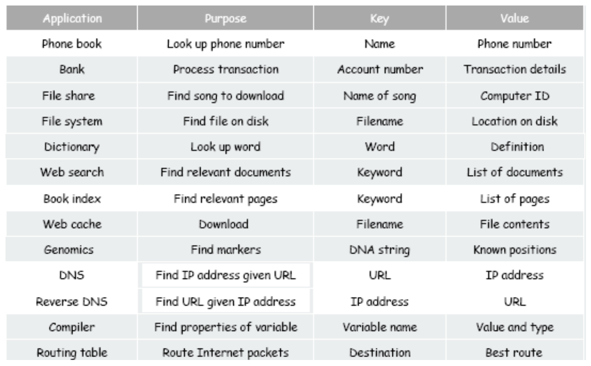
- Insert URL with specified IP address.
- Given URL, find corresponding IP address



URL	IP address
www.cs.princeton.edu	128.112.136.11
www.princeton.edu	128.112.128.15
www.yale.edu	130.132.143.21
www.harvard.edu	128.103.060.55
www.simpsons.com	209.052.165.60
key	value

Can interchange roles: given IP address find corresponding URL

Example applications



Application	Purpose	Key	Value
Phone book	Look up phone number	Name	Phone number
Bank	Process transaction	Account number	Transaction details
File share	Find song to download	Name of song	Computer ID
File system	Find file on disk	Filename	Location on disk
Dictionary	Look up word	Word	Definition
Web search	Find relevant documents	Keyword	List of documents
Book index	Find relevant pages	Keyword	List of pages
Web cache	Download	Filename	File contents
Genomics	Find markers	DNA string	Known positions
DNS	Find IP address given URL	URL	IP address
Reverse DNS	Find URL given IP address	IP address	URL
Compiler	Find properties of variable	Variable name	Value and type
Routing table	Route Internet packets	Destination	Best route

Implementation

- Define a structure to store key-value pairs

Example: phonebook

```
typedef struct {
   long number;
   char name[80]
} PhoneEntry;
```

The key is phone number and the value is person name

Using array for implementation

- Key-value pairs are stored in an ordered array

Example:

```
#define MAX_PHONE_NUMBER 1000
typedef struct {
   PhoneEntry entries[MAX_PHONE_NUMBER];
   int total;
} PhoneBook;
```

API

- Add an entry in the phone book

```
void addPhoneNumber(long number, char *
   name, PhoneBook* book);
```

NB: If the entry exists, the value should be overwritten.

- Find an entry in the phone book

```
char * getPhoneNumber(long number, const
   PhoneBook* book);
```

returns null if the entry does not exist

Quiz 1

- Write a program to add and search phone numbers in a phone book using an array for implementation

Using dynamic memory

- The memory to store the entries should be allocated dynamically according to the size of the phone book.

```
typedef struct {
   PhoneEntry * entries;
   int total;
   int size;
} PhoneBook;
```

When the total number of exceeds the size, the memory entries have to be reallocated with a new size

API

```
#define INITIAL_SIZE 100
#define INCREMENTAL_SIZE 10
```

- Create a phone book with an initial size

```
PhoneBook createPhoneBook();
```

- Drop a phone book

```
void dropPhoneBook(PhoneBook* book);
```

Quiz 2

- Rewrite the phone book program using dynamic memory

Generic symbol tables

- Define a generic structure for entries

```
typedef struct {
    void * key;
    void * value;
} Entry;
```

- Define a generic structure for symbol tables

```
typedef struct {
    Entry * entries;
    int size, total;
    Entry (makeNode*)(void*, void*);
    int (compare*)(void*, void*);
} SymbolTable;
```

makeNode is a function pointer to refer to a function to create a node with a key and a value passed

compare is a function to refer to a function to compare two keys

API

```
#define INITIAL_SIZE 100
#define INCREMENTAL_SIZE 10
SymbolTable createSymbolTable(
        Entry (makeNode*)(void*, void*),
        int (compare*)(void*, void*)
);
void dropSymbolTable(SymbolTable* tab);
int addEntry(void* key, void* value, SymbolTable* book);
void * getEntryValue(void* key, const SymbolTable *
    book);
```

NB: Free the memory allocated for each entry when a table is dropped

Example

```
Entry makePhoneBook(void* phone, void* name) {
    Entry res;
    res.key = malloc(sizeof(int));
    memcpy( res.key, phone, sizeof(int) );
    res.value = strdup( (char*)name );
    return res;
}
int comparePhone(void * key1, void* key2) {
    int num1 = *( (int*) key1 );
    int num2 = *( (int*) key2 );
    if (num1==num2) return 0;
    else if (num1 < num2) return -1;
    else return 1;
}

SymbolTable phoneBook = createSymbolTable(makePhoneBook,
    comparePhone);
```

Quiz 3

- Rewrite the phone book program using a generic symbol table

Homework

- Make a symbol table using a binary search tree and then use this data structure to write the phone book program.