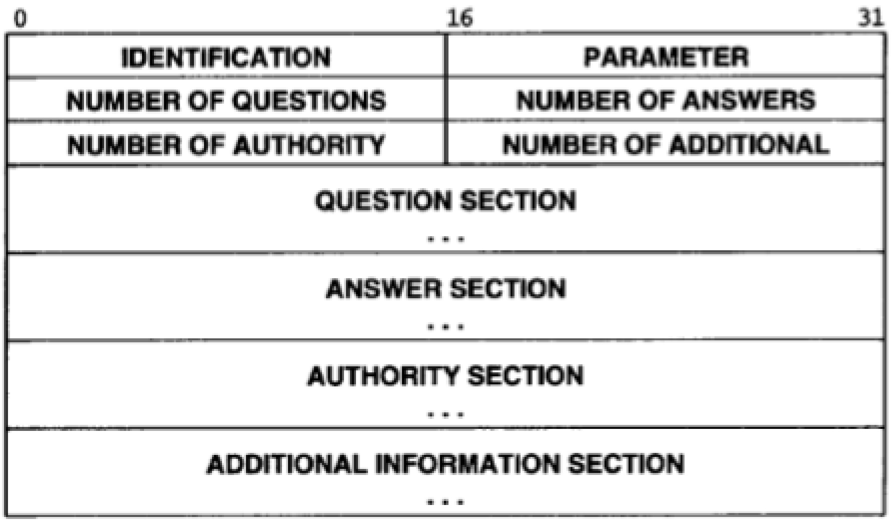
Implement DNS client to send Query to 8.8.8.8 with recursion bit on

SANCHIT KHANNA 2015A7PS0138P
MRIGESH MADAN 2015A7PS0146P



Each DNS message has a header. Components of the header are

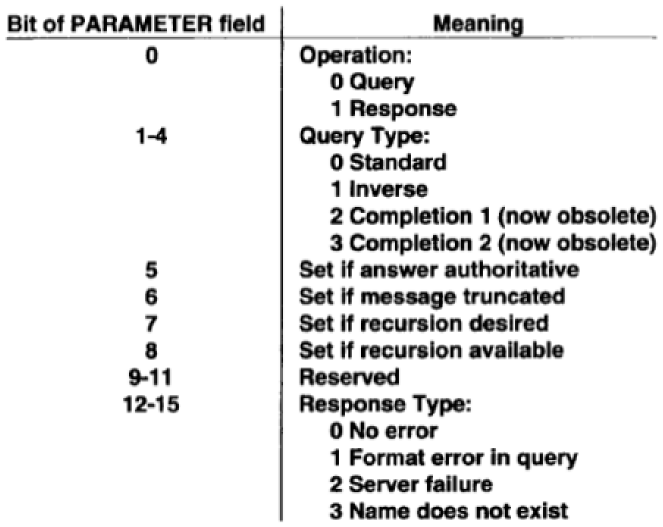
1)	IDENTIFICATION	2 BYTES	0 – 1
2)	PARAMETER	2 BYTES	2 – 3
3)	NUMBER OF QUESTIONS	2 BYTES	4 – 5
4)	NUMBER OF ANSWERS	2 BYTES	6 – 7
5)	NUMBER OF ATHORITY	2 BYTES	8 – 9
6)	NUMBER OF ADDITIONAL	2 BYTES	10-11
7)	QUESTION SECTION	(Arbitrary Length)	
8)	ANSWER SECTION	(Arbitrary Length)	
9)	AUTHORITY SECTION	(Arbitrary Length)	
10)	ADDITIONAL INFORMATION SECTION	(Arbitrary Length)	

Components (1-7) and filled as part of the DNS querry and Components (8-10) are returned as part of the answer to the request.

1) IDENTIFICATION

Field used by client to match the response to the querry.

2) PARAMETERS



Bit of PARAMETER field	Meaning
0	Operation: 0 Query 1 Response
1-4	Query Type: 0 Standard 1 Inverse 2 Completion 1 (now obsolete) 3 Completion 2 (now obsolete)
5	Set if answer authoritative
6	Set if message truncated
7	Set if recursion desired
8	Set if recursion available
9-11	Reserved
12-15	Response Type: 0 No error 1 Format error in query 2 Server failure 3 Name does not exist

Bit 7 of the PARAMETER field is set to make the query recursive.

3) NUMBER OF QUESTIONS

We are sending one DNS query at a time thus the number of questions is set to 1.

4) NUMBER OF ANSWERS

This field is set to 0 when querying. When the response to the query is receiver this field is populated by the server indicating the number of answers in response.

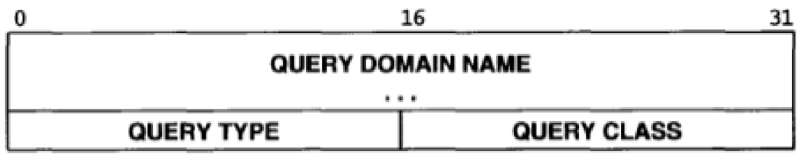
5) NUMBER OF AUTHORITY

This field is set to 0 when querying. When the response to the query is receiver this field is populated by the server indicating the number of answers in response.

6) NUMBER OF ADDITIONAL

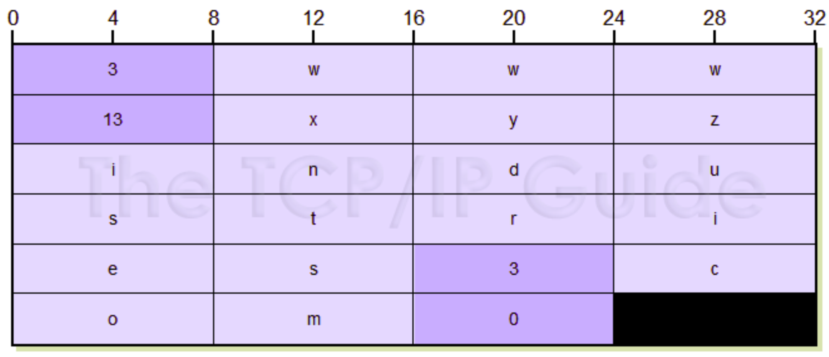
This field is set to 0 when querying. When the response to the query is receiver this field is populated by the server indicating the number of answers in response.

7) QUESTION SECTION



a) QUERY DOMAIN NAME

This field has variable length. Domain names are stored as a sequence of labels.
For Example:
www.xyzindustries.com is encoded as : [3]www[13]xyzindustries[3]com[0]



DNS uses specialo notation for DNS names. Each label is encoded one after the another, before each label a single byte is stored holding a binary number indicating the number of characters in the label. Thus length of the label can be max 63 characters. The end of the name is indicated by a null label ("0").

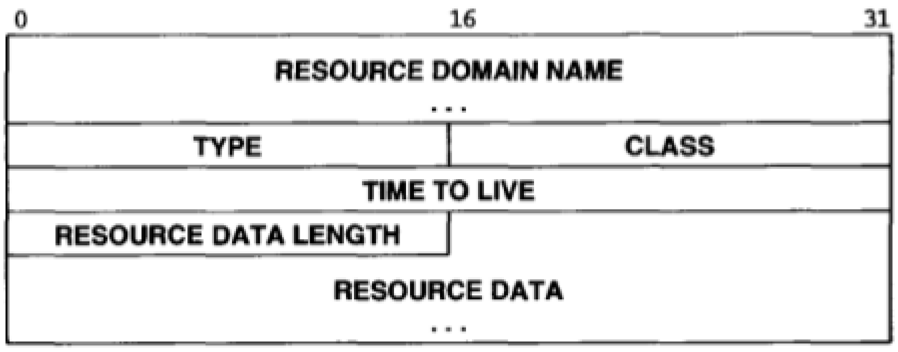
b) QUERY TYPE

It encodes the type of question. The equivalent QTYPE 16 bit values can be found at https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-4

c) QUERY CLASS

It allows domain names to be used for arbitrary objects, here Internet(1) is only one possible class.
https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-4

8) ANSWER SECTION



a) RESOURCE DOMAIN NAME

It is encoded same as Query Domain Name having arbitrary length.

b) TYPE (2 BYTES)

It includes the type of data included in the resource record.(Same as Query Type)

c) CLASS (2 BYTES)

It specifies the data Class.(Same as Query Class)

d) TIME TO LIVE (4 BYTES)

32 Bit integer that specifies the number of seconds information in this resource record has to be cashed.

e) RESOURCE DATA LENGTH (2 BYTES)

It stores the length of the Resource Data Field

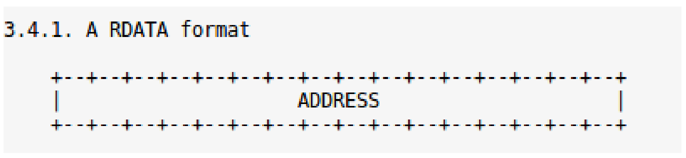
f) RESOURCE DATA

- **A Record Type** (32 Bit value)

  in_addr structure is used to store the 32 bit IPV4 address.

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    in_port_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};

/* Internet address. */
struct in_addr {
    uint32_t       s_addr;     /* address in network byte order */
};
```

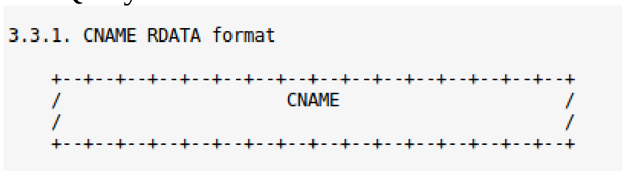


```
3.4.1. A RDATA format

    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    ADDRESS                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

- **CNAME Record Type** (Arbitrary Length record)

  Stored same as Query Domain Name.

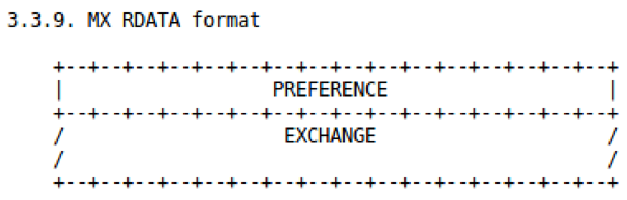


```
3.3.1. CNAME RDATA format

    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                     CNAME                     /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

- **MX Record Type** (2 Byte for PREFERENCE and Arbitrary Length EXCHANGE)

  Exchange is stored same as Query Domain Name



```
3.3.9. MX RDATA format

    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                  PREFERENCE                   |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                   EXCHANGE                    /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

- **PTR Record Type** (Arbitrary Length Record)
  Stored Same as Query Domain Name

```
3.3.12. PTR RDATA format

    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                   PTRDNAME                    /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

- **AAAA Record Type** (128 Bit Value)
  in6_addr structure is used to store the 128 bit IPV6 address.

```
struct sockaddr_in6 {
    sa_family_t     sin6_family;   /* AF_INET6 */
    in_port_t       sin6_port;     /* port number */
    uint32_t        sin6_flowinfo; /* IPv6 flow information */
    struct in6_addr sin6_addr;     /* IPv6 address */
    uint32_t        sin6_scope_id; /* Scope ID (new in 2.4) */
};

struct in6_addr {
    unsigned char   s6_addr[16];   /* IPv6 address */
};
```

- **Other Records Types**
  RDATA Records can be found at http://www.ietf.org/rfc/rfc1035.txt

9) AUTHORITY SECTION

Stored same as Answer Section.

10) ADDITIONAL INFORMATION SECTION

Stored same as Answer Section.

DNS Name Notation and Message Compression Techniques

To cut down on duplication, a special technique called message compression is used. Instead of a DNS name encoded as above using the combination of labels and label-lengths, a two-byte subfield is used to represent a pointer to another location in the message where the name can be found. The first two bits of this subfield are set to one (the value “11” in binary), and the remaining 14 bits contain an offset that species where in the message the name can be found, counting the first byte of the message (the first byte of the ID field) as 0.

Let's go back to our example. Suppose that in the DNS message above, the RData field of the MX record, containing “mail.xyzindustries.com”, begins at byte 47. In this first instance, we would find the name encoded in full as: [3]www[13]xyzindustries[3]com[0].

However, the second instance, where “mail.xyzindustries.com” shows up in the Name field of the A record, we would instead put two “1” bits, followed by the number 47 encoded in binary. So, this would be the 16-bit binary pattern “11000000 00101111”, or two numeric byte values “192” and “47”. This second instance now takes 2 bytes instead of duplicating the 24 bytes needed for the first instance of the name.

How does a device reading a name field differentiate a pointer from a “real” name? This is the reason that “11” is used at the start of the field. Doing this guarantees that the first byte of the pointer will always have a value of 192 or larger. Since labels are restricted to a length of 63 or less, when the host reads the first byte of a name, if it sees a value of 63 or less in a byte, it knows this is a “real” name; a value of 192 or more means it is a pointer.

void printfunc(char *offstr,char *start) makes use of the Compression Technique using recursive calls till a [0] is encontered in the domain name. It recurses when it finds a byte with value 0xC0 which is followed by the offset.

void dg_cli(char*, int, struct sockaddr_in*,socklen_t, int);
This function is used to fill components (1-7) of the DNS Query in the same format specified earlier. It keeps track of the offset where the QUESTION SECTION ends. Answer of the query is parsed from this offset.

int parse(char *recvline, int answer_offset,int qtype);

- This function checks all the PARAMETER Bits (Authoritative Answer, Truncated, Recursion Desired, Recursion Available and Response Code).
- It displays the number of Questions, Anwers, authoritative answers and additional records
- According to the Querry Type the Resource Data is parsed and printed.

(it uses printfunc() to print records having arbitrary length while A and AAAA records are printed by typecasting into the respective Structures mentioned earlier)

int main(int argc, char* argv[]);
This function is the main driver funtion, which reads the DNS IP to be querried from the “resolv.conf” file and calls the dg_cli() function to form the DNS query.

RUNNING THE PROGRAM

```
./<Executable_Name> <Domain_Name> <Query_Type>
```

Example

```
./dns_client www.google.com A
./dns_client www.google.com AAAA
./dns_client mail.google.com MX
./dns_client mail.google.com PTR
./dns_client mail.google.com CNAME
```