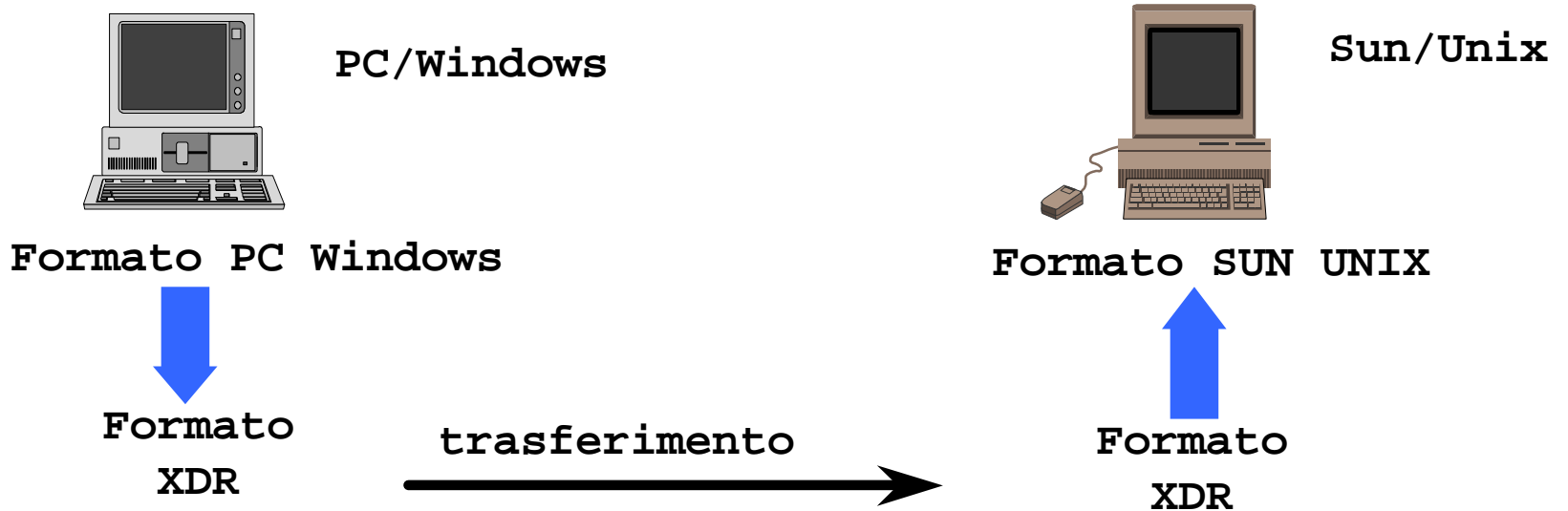


XDR (eXternal Data Representation)

- E` uno standard per la descrizione e la codifica dei dati (rappresentazione aperta per lo scambio di dati tra piattaforme eterogenee)
- Originariamente introdotto dalla SUN
- Standardizzato come Rfc (rfc 1014, rfc 1832)
- Svolge un ruolo simile a quello di altri linguaggi per la descrizione astratta dei dati (ASN.1)

Tipico uso di XDR

- XDR è utile come linguaggio comune per trasferire dati tra macchine con architetture e/o sistemi operativi diversi



Concetti basilari

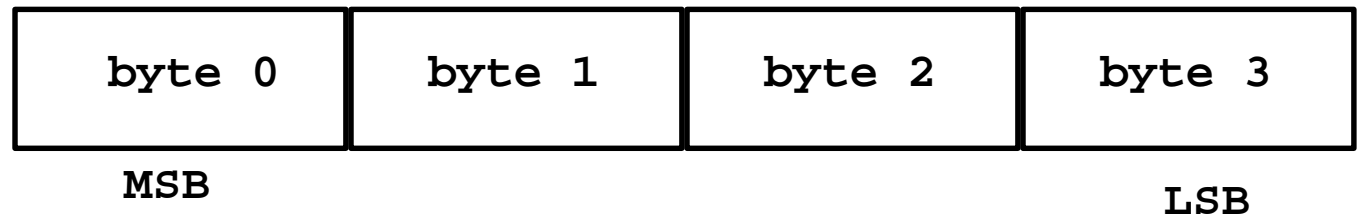
- XDR assume che il byte sia l'unità **portabile** (non ci siano ambiguità sull'interpretazione dei bit)
- La parola di 4 byte è l'unità fondamentale di tutte le rappresentazioni XDR (compromesso per minimizzare gli allineamenti e la dimensione dei messaggi)
- Lo standard XDR definisce:
 - **Un linguaggio C-like per descrivere i tipi di dato (solo i dati!)**
 - **Una rappresentazione univoca per ciascun dato descritto** (sequenza ordinata di byte, dove il primo byte - o byte 0 - è il primo ad essere trasferito/memorizzato)

Interi con segno

- Sintassi:

int *identifier*

- Rappresentazione: (big endian)



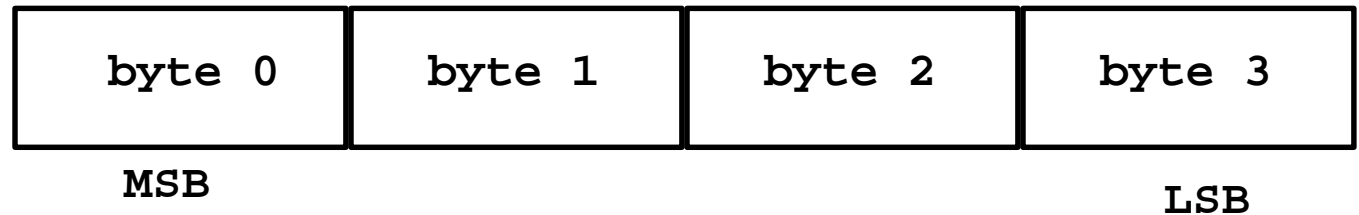
- Codifica: Complemento a 2

Interi senza segno

- Sintassi:

unsigned int *identifier*

- Rappresentazione:



- Codifica: Binario puro

Dati definiti per enumerazione

- Sintassi:

enum { *name_id = const* , ... } *identifier*

- Rappresentazione e Codifica: quella degli interi con segno (sono legali però solo i valori definiti)

- Esempio:

enum {BIANCO=0, NERO=1, ROSSO=1, BLU=2} colore

Dati booleani

- Sintassi:

bool *identifier*

- Equivale a:

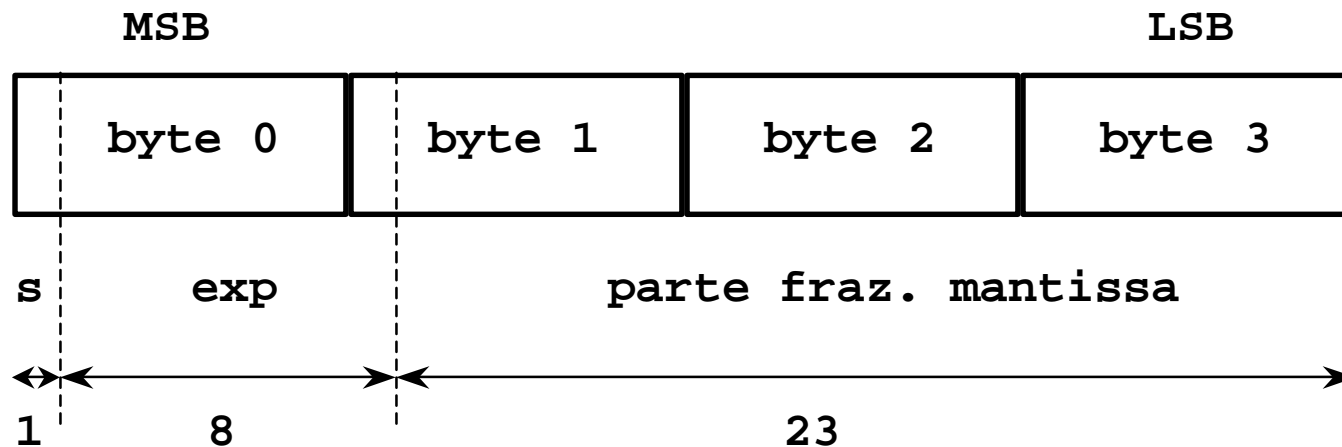
– enum {FALSE=0, TRUE=1 } *identifier*

Numeri floating point

- Sintassi:

float *identifier*

- Rappresentazione e Codifica: IEEE 754 s.p.



Rappresentazioni estese

- Interi su 8 byte (complemento a 2)

hyper *identifier*

unsigned hyper *identifier*

- Floating point doppia precisione (IEEE 754 d.p. - 8 byte): 1 segno, 11 esponente, 52 mantissa:

double *identifier*

- Floating point quadrupla precisione (16 byte): 1 segno, 15 esponente, 112 mantissa:

quadruple *identifier*

Void

- È l'elemento nullo
- Sintassi:

void

- Rappresentazione e Codifica:

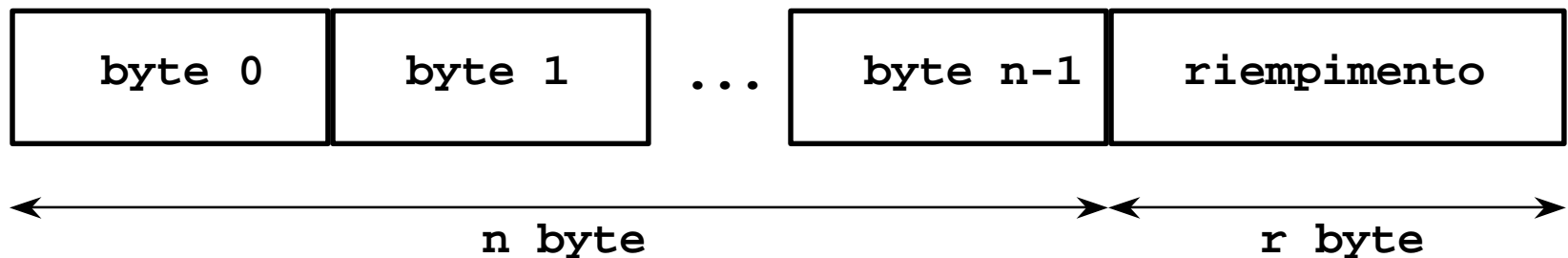
Il void è rappresentato su 0 byte.

Dati “opachi” a lunghezza fissa

- Sono sequenze di byte di cui non sono rilevanti il significato e la codifica.
- Sintassi:

opaque *identifier* [n]

- Rappresentazione:



r varia da 0 a 3 $((n+r) \bmod 4 = 0)$

Dati “opachi” a lunghezza variabile

- Sintassi:

opaque *identifier* <*n*> (n è la lunghezza max)

opaque *identifier* <> (lungh. max = $2^{32}-1$)

- Rappresentazione:



La lunghezza effettiva L è rappresentata come int

Il resto è rappresentato come la sequenza a lungh. fissa

Stringhe

- Sono rappresentate come i dati opachi a lunghezza variabile

- Sintassi:

string *identifier* <*n*> (n è la lunghezza max)

string *identifier* <> (lungh. max = $2^{32}-1$)

- Rappresentazione: Il byte 4 (il primo dopo la lunghezza) è il primo carattere della stringa.
- Codifica: ASCII su 1 byte

Array a dimensione fissa

- Sono sequenze di dati di tipo omogeneo (ciascuno rappresentato su un multiplo di 4 byte)

- Sintassi:

tipo *identifier* [*n*]

- Rappresentazione:

Semplice sequenza delle singole rappresentazioni (gli elementi possono avere dimensioni diverse!)

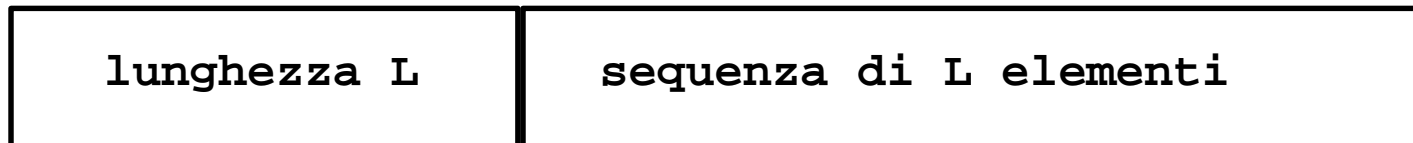
Array a dimensione variabile

- Sintassi:

tipo *identifier* <n> (lungh. max=n)

tipo *identifier* <> (lungh. max = $2^{32}-1$)

- Rappresentazione:



La lunghezza effettiva L è rappresentata come int

Il resto è rappresentato come l'array a dimensione fissa

Strutture

- Sono sequenze di dati di tipo diverso (ciascuno rappresentato su un multiplo di 4 byte)
- Sintassi:

```
struct { elemento_1 ;  
        ...  
        elemento_n ; }      identifier
```

- Rappresentazione:

Semplice sequenza delle singole rappresentazioni

Union

- Sono costituite da un elemento discriminatore, seguito da un elemento il cui tipo dipende dal valore del discriminatore
- Sintassi:

```
union switch( discriminatore ) {  
  case valore_1 : elemento_1 ;  
  ...  
  case valore_n : elemento_n ;  
  default      : elemento_def ; }   identifier
```

Union (cont)

- Il discriminatore è di tipo int o unsigned int o enum
- Il ramo default è opzionale
- Rappresentazione:



← 4 byte →

l'elemento rappresentato dopo il discriminatore è quello del ramo case relativo al valore del discriminatore.

Costanti e Typedef

- È possibile assegnare nomi simbolici a valori e a tipi di dato:
- Sintassi per le costanti:

const *identifier = value*

- Sintassi per i tipi (come in linguaggio C):

typedef *declaration*

oppure, per struct, union o enum, si può usare la forma:

```
enum bool { FALSE = 0;  
           TRUE  = 1; }
```

Elemento opzionale

- Sintassi:

*type * identifier*

- Equivale ad un array di dimensione variabile, la cui dimensione può essere solo 0 oppure 1:

type identifier<1>

- Rappresentazione:



← 4 byte →

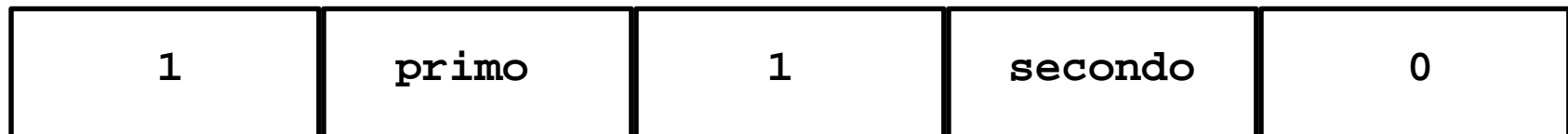
Liste

- Vengono definite recursivamente usando i typedef e gli elementi opzionali.

- Esempio: lista di interi

```
struct * intlist {  
    int    item;  
    intlist next;  
}
```

- Rappresentazione di una lista di 2 interi:



Esempio: un semplice tipo XDR per rappresentare un file

```
/* Definitions of constants:
*/
const MAXUSERNAME = 32;      /* max length of a user name */
const MAXFILELEN = 65535;   /* max length of a file      */
const MAXNAMELEN = 255;    /* max length of a file name */

/* Types of files:
*/
enum filekind {
    TEXT = 0,                /* ascii data */
    DATA = 1,               /* raw data   */
    EXEC = 2                  /* executable */
};
```

Esempio: un semplice tipo XDR per rappresentare un file (II)

```
/* File information, per kind of file:
*/
union filetype switch (filekind kind) {
    case TEXT: void;          /* no extra information */
    case DATA:
        string creator<MAXNAMELEN>; /* data creator */
    case EXEC:
        string interpretor<MAXNAMELEN>; /* program interpretor */
};

/* A complete file:
*/
struct file {
    string filename<MAXNAMELEN>; /* name of file */
    filetype type;                /* info about file */
    string owner<MAXUSERNAME>;   /* owner of file */
    opaque data<MAXFILELEN>;     /* file data */
};
```

Libreria XDR

- La manipolazione dei dati in formato XDR viene realizzata da un'apposita libreria (normalmente disponibile sui sistemi UNIX)
- Tipiche operazioni:
 - conversioni da/verso tipi C
- La libreria normalmente usa il *buffer paradigm*:
 - il dato XDR viene costruito/memorizzato in un buffer
- Esistono strumenti per generare automaticamente il codice di codifica/decodifica di tipi XDR complessi

Tipica Codifica XDR

```
#include <rpc/xdr.h>
#define BUFSIZE 4000
XDR * xdrs;          /* pointer to XDR stream */
char buf[BUFSIZE]; /* buffer for XDR data */

/* create stream */
xdrmem_create(xdrs, buf, BUFSIZE, XDR_ENCODE);

/* append data to XDR stream */
...
xdr_int(xdrs, &i); /* convert/append integer i */
...

/* destroy (free) stream */
xdr_destroy(xdrs);
```

Tipica Decodifica XDR

```
#include <rpc/xdr.h>
#define BUFSIZE 4000
XDR * xdrs;          /* pointer to XDR stream */
char buf[BUFSIZE]; /* buffer for XDR data */

/* create stream */
xdrmem_create(xdrs, buf, BUFSIZE, XDR_DECODE);

/* extract data from XDR stream */
...
xdr_int(xdrs, &i); /* extract/convert integer i */
...

/* destroy (free) stream */
xdr_destroy(xdrs);
```

Esempio: i messaggi RPC

```
struct rpc_msg {
    unsigned int xid;
    union switch (msg_type mtype) {
        case CALL:
            call_body cbody;
        case REPLY:
            reply_body rbody;
    } body;
};
```

```
enum msg_type {
    CALL = 0,
    REPLY = 1
};
```

Corpo della Chiamata

```
struct call_body {
    unsigned int rpcvers; /* must be equal to 2 */
    unsigned int prog;
    unsigned int vers;
    unsigned int proc;
    opaque_auth cred;
    opaque_auth verf;
    /* procedure specific parameters start here */
};
```

Campo Autenticazione

```
enum auth_flavor {
    AUTH_NULL          = 0,
    AUTH_UNIX          = 1,
    AUTH_SHORT          = 2,
    AUTH_DES            = 3
    /* and more to be defined */
};
```

```
struct opaque_auth {
    auth_flavor flavor;
    opaque body<400>;
};
```

Corpo della Risposta

```
union reply_body switch (reply_stat stat) {  
    case MSG_ACCEPTED:  
        accepted_reply areply;  
    case MSG_DENIED:  
        rejected_reply rreply;  
} reply;
```

```
enum reply_stat {  
    MSG_ACCEPTED = 0,  
    MSG_DENIED   = 1  
};
```

Chiamata accettata

```
struct accepted_reply {
    opaque_auth verf;
    union switch (accept_stat stat) {
    case SUCCESS:
        opaque results[0];
        /*procedure-specific results start here
        */
    case PROG_MISMATCH:
        struct {
            unsigned int low;
            unsigned int high;
        } mismatch_info;
    default:
        /* Void.  Cases include PROG_UNAVAIL,
        PROC_UNAVAIL, and GARBAGE_ARGS.
        */
        void;
    } reply_data;
};
```

```
enum accept_stat {
    SUCCESS          = 0, /* RPC executed successfully      */
    PROG_UNAVAIL     = 1, /* remote hasn't exported program */
    PROG_MISMATCH    = 2, /* remote can't support version # */
    PROC_UNAVAIL     = 3, /* program can't support procedure */
    GARBAGE_ARGS     = 4  /* procedure can't decode params  */
};
```

Chiamata rifiutata

```
union rejected_reply switch (reject_stat stat) {
    case RPC_MISMATCH:
        struct {
            unsigned int low;
            unsigned int high;
        } mismatch_info;
    case AUTH_ERROR:
        auth_stat stat;
};

enum reject_stat {
    RPC_MISMATCH=0, /* RPC version number != 2          */
    AUTH_ERROR=1   /* remote can't authenticate caller */
};
```