Programmazione B
Ingegneria e Scienze Informatiche - Cesena
A.A. 2021-2022

Le stringhe

Andrea Piroddi - andrea.piroddi@unibo.it
CREDIT: Pietro Di Lena

Two strings walk into a bar and sit down.
The bartender says, "So what'll it be?". The first string says,
"I think I'll have a beerjfdLk jk3s d#f67howe%^U r89nvyowmc63^Dz x.xvcu"
"Please excuse my friend.", the second string says. "He isn't null-terminated."

Stringhe

- In C non esiste un tipo di dato specifico per le stringhe.
- Una stringa è un array monodimensionale di `char`, *terminato* dal **carattere nullo** `'\0'`.
- La presenza del carattere nullo `'\0'` permette di fare distinzione tra gli array di `char` **generici** e gli array di `char` che possono essere trattati come **stringhe**.

Stringhe

- In C non esiste un tipo di dato specifico per le stringhe.
- Una stringa è un array monodimensionale di `char`, *terminato* dal **carattere nullo** `'\0'`.
- La presenza del carattere nullo `'\0'` permette di fare distinzione tra gli array di `char` **generici** e gli array di `char` che possono essere trattati come **stringhe**.
- Ad esempio, lo specificatore di formato per le stringhe `%s` della `printf()` produce un output non definito se l'argomento corrispondente è un array di `char` non contenente almeno una occorrenza del carattere nullo.
- Quando si dichiara un array di `char` per memorizzare una stringa di lunghezza `N` è quindi sempre necessario prevedere una cella di memoria in più per la memorizzazione del carattere nullo.
- Il codice ascii del carattere nullo `'\0'` è 0.

Dichiarazione ed inizializzazione di stringhe

- Abbiamo due modi possibili per inizializzare stringhe, dichiarate come array di `char`: assegniamo i *singoli caratteri* oppure assegniamo una costante *stringa letterale*.

```
1  char s1[7] = {'s','t','r','i','n','g','\0'};
2  char s2[6] = {'s','t','r','i','n','g'};
3  char s3[9] = {'s','t','r','i','n','g'};
4  char s4[ ] = {'s','t','r','i','n','g'};
5  char s5[ ] = {'s','t','r','i','n','g','\0'};
6  char s6[6] = "string";
7  char s7[ ] = "string";
8
9  printf("%s\n",s1); // OK
10 printf("%s\n",s2); // Non definita
11 printf("%s\n",s3); // OK
12 printf("%s\n",s4); // Non definita
13 printf("%s\n",s5); // OK
14 printf("%s\n",s6); // Non definita
15 printf("%s\n",s7); // OK
```

- Gli array `s2[]` e `s6[]` non hanno spazio sufficiente per il carattere nullo. Lo stesso vale per `s4[]`, la cui lunghezza viene determinata dall'inizializzatore.
- In `s3[]` gli elementi indicizzati da 6 a 8 sono automaticamente inizializzati a zero, che corrisponde a `'\0'`. Non è quindi necessario specificare `'\0'` nell'inizializzazione di `s1[]` mentre è necessario farlo per `s5[]`, dato che non indichiamo la sua lunghezza.
- L'inizializzazione di `s7[]` non presenta problemi dato che la costante stringa contiene implicitamente il carattere nullo dopo il carattere `'g'`.

Carattere nullo e stringhe letterali: ulteriori precisazioni

- La prima occorrenza di un carattere nullo in un array di `char` indica la fine della stringa, indipendentemente dal contenuto delle celle successive.

```
1 char s1[] = {'s','t','r','\0','i','n','g','\0'};
2 char s2[] = {'\0','s','t','r','i','n','g'};
3
4 printf("%s\n",s1); // Stampa "str"
5 printf("%s\n",s2); // Stampa stringa vuota
```

Carattere nullo e stringhe letterali: ulteriori precisazioni

- La prima occorrenza di un carattere nullo in un array di `char` indica la fine della stringa, indipendentemente dal contenuto delle celle successive.

```
1 char s1[] = {'s','t','r','\0','i','n','g','\0'};
2 char s2[] = {'\0','s','t','r','i','n','g'};
3
4 printf("%s\n",s1); // Stampa "str"
5 printf("%s\n",s2); // Stampa stringa vuota
```

- Possiamo *concatenare* stringhe letterali distinte indicandole di seguito, una dopo l'altra.

```
1 char s1[] = "str" "ing";
2 char s2[] = "s" "t" "r" "i" "n" "g";
3 char s3[] = "string";
4
5 printf("%s\n",s1); // Stampa "string"
6 printf("%s\n",s2); // Stampa "string"
7 printf("%s\n",s3); // Stampa "string"
```

  Le inizializzazioni a riga 1 e 2 sono equivalenti a quella su riga 3. Nella concatenazione di stringhe letterali un unico carattere nullo viene posizionato a fine stringa.
- Le stringhe sono effettivamente array di `char`, quindi le inizializzazioni con lista di caratteri o tramite stringhe letterali sono possibili solo al momento della dichiarazione.

La libreria ctype

- Il linguaggio C mette a disposizione librerie per lavorare con caratteri e stringhe.
- L'header `ctype.h` dichiara diverse funzioni utili per *verificare* proprietà e `convertire` i tipi di dato `char`.
- Sono indicate per scrivere codice portabile che non dipenda dalla rappresentazione dei caratteri (tipicamente ascii) su di un particolare tipo di piattaforma.

La libreria ctype

- Il linguaggio C mette a disposizione librerie per lavorare con caratteri e stringhe.
- L'header `ctype.h` dichiara diverse funzioni utili per *verificare* proprietà e `convertire` i tipi di dato `char`.
- Sono indicate per scrivere codice portabile che non dipenda dalla rappresentazione dei caratteri (tipicamente ascii) su di un particolare tipo di piattaforma.
- I prototipi delle funzioni in `ctype.h` e la loro semantica è definita dallo standard ISO.
  - Tutte le funzioni prendono in input un tipo di dato `int`, il cui valore possa essere rappresentato con un `unsigned char`, oppure sia equivalente al valore di `EOF`.
  - Se l'argomento assume un qualsiasi altro valore, allora il loro comportamento è non definito.

La libreria ctype

- Il linguaggio C mette a disposizione librerie per lavorare con caratteri e stringhe.
- L'header `ctype.h` dichiara diverse funzioni utili per *verificare* proprietà e `convertire` i tipi di dato `char`.
- Sono indicate per scrivere codice portabile che non dipenda dalla rappresentazione dei caratteri (tipicamente ascii) su di un particolare tipo di piattaforma.
- I prototipi delle funzioni in `ctype.h` e la loro semantica è definita dallo standard ISO.
  - Tutte le funzioni prendono in input un tipo di dato `int`, il cui valore possa essere rappresentato con un `unsigned char`, oppure sia equivalente al valore di `EOF`.
  - Se l'argomento assume un qualsiasi altro valore, allora il loro comportamento è non definito.
- Ricordiamo che il tipo di dato `char` può essere di default `signed` o `unsigned`, a seconda dell'implementazione.
- Quando il tipo di default di `char` è `signed char` allora il comportamento delle funzioni potrebbe non essere definito se il valore della variabile `char` è negativo.
- Una soluzione per evitare problemi è quella di effettuare un casting a `unsigned char` con parametri di tipo `char`.

Funzioni nella libreria ctype

Classe	Prototipo	Descrizione
Funzioni di verifica	`int isalnum(int c);`	Verifica che il carattere sia *alfanumerico*
	`int isalpha(int c);`	Verifica che il carattere sia *alfabetico*
	`int iscntrl(int c);`	Verifica che il carattere sia di *controllo*
	`int isdigit(int c);`	Verifica che il carattere sia *numerico*
	`int isgraph(int c);`	Verifica che il carattere sia *grafico* (carattere stampabile, eccetto spazio)
	`int islower(int c);`	Verifica che il carattere sia *minuscolo*
	`int isprint(int c);`	Verifica che il carattere sia *stampabile* (incluso spazio)
	`int ispunct(int c);`	Verifica che il carattere sia di *spaziatura*
	`int isspace(int c);`	Verifica che il carattere sia di *spaziatura*
	`int isupper(int c);`	Verifica che il carattere sia *maiuscolo*
	`int isxdigit(int c);`	Verifica che il carattere sia *esadecimale*
Funzioni di conversione	`int tolower(int c);`	Converte in minuscolo (se applicabile)
	`int toupper(int c);`	Converte in maiuscolo (se applicabile)

- Le funzioni di verifica ritornano un valore diverso da zero (true) se il test effettuato è positivo.
- Le funzioni di conversione ritornano il codice ascii del carattere convertito, diversamente il valore passato in input.

Funzioni nella libreria ctype: esempi

- Come ricodificare una stringa in minuscolo o maiuscolo.

```
1  // Converte i caratteri di una stringa in minuscolo
2  void tolow(char s[]) {
3    int i = 0;
4    while(s[i] = tolower((unsigned char)s[i])) i++;
5  }
6
7  // Converte i caratteri di una stringa in maiuscolo
8  void toup(char s[]) {
9    int i = 0;
10   while(s[i] = toupper((unsigned char)s[i])) i++;
11 }
```

Le due funzioni assumono in input una stringa (terminata da '\0'). Diversamente il loro comportamento è non definito. Da notare la condizione di stop del `while`.

Funzioni nella libreria ctype: esempi

- Come ricodificare una stringa in minuscolo o maiuscolo.

```
1  // Converte i caratteri di una stringa in minuscolo
2  void tolow(char s[]) {
3    int i = 0;
4    while(s[i] = tolower((unsigned char)s[i])) i++;
5  }
6
7  // Converte i caratteri di una stringa in maiuscolo
8  void toup(char s[]) {
9    int i = 0;
10   while(s[i] = toupper((unsigned char)s[i])) i++;
11 }
```

Le due funzioni assumono in input una stringa (terminata da '\0'). Diversamente il loro comportamento è non definito. Da notare la condizione di stop del `while`.

- Come contare il numero di caratteri alfanumerici in una stringa (lettere e cifre).

```
1  // Conta il numero di caratteri alfanumerici in una stringa
2  unsigned int count_alnum(char s[]) {
3    unsigned int i = 0, c = 0;
4    while(s[i])
5      if(isalnum((unsigned char)s[i++])) c++;
6    return c;
7  }
```

Come prima, funziona solo con array di `char` di tipo stringa.

La libreria `string`

- L'unico effettivo supporto che il linguaggio C fornisce per la manipolazione delle stringhe è la gestione delle stringhe letterali.
- La libreria standard del linguaggio C mette a disposizione una serie di funzioni per manipolare array di `char` terminati dal carattere nullo.
- I prototipi di tali funzioni sono specificati nel file header `string.h`.

La libreria `string`

- L'unico effettivo supporto che il linguaggio C fornisce per la manipolazione delle stringhe è la gestione delle stringhe letterali.
- La libreria standard del linguaggio C mette a disposizione una serie di funzioni per manipolare array di `char` terminati dal carattere nullo.
- I prototipi di tali funzioni sono specificati nel file header `string.h`.
- Le dichiarazioni dei prototipi in `string.h` contengono nuovi elementi, che abbiamo solo accennato finora ma non ancora visto nel dettaglio. In particolare:
  - **Puntatori**. Abbiamo già visto che passare un array ad una funzione significa passare alla funzione l'indirizzo di memoria del primo elemento dell'array. Le funzioni in `string.h` dichiarano esplicitamente indirizzi di memoria come parametri, facendo uso della notazione puntatore.
  - **Lunghezze**. I parametri relativi alle lunghezze delle stringhe sono di tipo `size_t`, che potrebbe essere, ad esempio, `unsigned int` o `unsigned long int` (dipende dall'implementazione).

La libreria `string`

- L'unico effettivo supporto che il linguaggio C fornisce per la manipolazione delle stringhe è la gestione delle stringhe letterali.
- La libreria standard del linguaggio C mette a disposizione una serie di funzioni per manipolare array di `char` terminati dal carattere nullo.
- I prototipi di tali funzioni sono specificati nel file header `string.h`.
- Le dichiarazioni dei prototipi in `string.h` contengono nuovi elementi, che abbiamo solo accennato finora ma non ancora visto nel dettaglio. In particolare:
  - **Puntatori**. Abbiamo già visto che passare un array ad una funzione significa passare alla funzione l'indirizzo di memoria del primo elemento dell'array. Le funzioni in `string.h` dichiarano esplicitamente indirizzi di memoria come parametri, facendo uso della notazione puntatore.
  - **Lunghezze**. I parametri relativi alle lunghezze delle stringhe sono di tipo `size_t`, che potrebbe essere, ad esempio, `unsigned int` o `unsigned long int` (dipende dall'implementazione).
- Se l'indirizzo passato ad una funzione nella libreria `string` punta ad una locazione di memoria al di fuori dell'array o se l'array (di `char`) non contiene un carattere nullo, il comportamento della funzioni è non definito.
- Diverse funzioni di libreria in `string.h` possono causare buffer overflow, se non utilizzate correttamente.

Alcune funzioni nella libreria `string`

Classe	Prototipo	Descrizione
Funzioni di copia	`char *strcpy(char *s1, const char *s2);` `char *strncpy(char *s1, const char *s2, size_t n);`	Copia i caratteri di `s2` in `s1`, incluso `'\0'`. Copia massimo `n` caratteri di `s2` in `s1`.
Funzioni di concatenazione	`char *strcat(char *s1, const char *s2);` `char *strncat(char *s1, const char *s2, size_t n);`	Aggiunge i caratteri di `s2` a `s1`. Aggiunge massimo `n` caratteri di `s2` a `s1`.
Funzioni di confronto	`int strcmp(const char *s1, const char *s2);` `int strncmp(const char *s1, const char *s2, size_t n);`	Confronta lessicograficamente `s1` e `s2`. Confronta massimo `n` caratteri di `s1` e `s2`.
Funzioni di ricerca	`char *strchr(const char *s, int c);` `char *strrchr(const char *s, int c);`	Individua la prima occorrenza di `c` in `s`. Individua l'ultima occorrenza di `c` in `s`.
Funzioni miscellanee	`size_t strlen(const char *s);`	Ritorna la lunghezza della stringa `s`.

- Le funzioni di copia e concatenazione ritornano il valore `s1`, cioè l'indirizzo di memoria della stringa `s1`.
- Le funzioni di confronto ritornano un valore intero:
  - se il valore è 0, allora i caratteri confrontati coincidono perfettamente,
  - se il valore è < 0, allora la stringa `s1` precede `s2` in ordine lessicografico,
  - se il valore è > 0, allora la stringa `s2` precede `s1` in ordine lessicografico.
- Le funzioni di ricerca ritornano l'indirizzo di memoria del carattere/stringa cercato, oppure `NULL`.
- Le funzioni `strcpy()` e `strcat()` non sono sicure e non dovrebbero essere utilizzate: perché?

Funzioni nella libreria string: esempi 1/3

- Come usare le funzioni di copia e confronto

```c
#include <stdio.h>
#include <string.h>

#define N 10

int main() {
  char s1[N+1], s2[N+1];
  int res;

  // Copia massimo N caratteri delle stringhe costanti negli array
  strncpy(s1,"a",N);
  strncpy(s2,"zuzzurellone",N);

  // Confronta le stringhe
  res = strcmp(s1,s2);

  if(res < 0)        // s1 precede lessicograficamente s2
    printf("%s < %s\n",s1,s2);
  else if (res > 0) // s2 precede lessicograficamente s1
    printf("%s > %s\n",s2,s1);
  else               // s1 e s2 sono uguali
    printf("%s = %s\n",s1,s2);

  return 0;
}
```

Funzioni nella libreria string: esempi 2/3

- Come usare le funzioni di copia e concatenazione

```c
#include <stdio.h>
#include <string.h>

#define N 50

int main() {
  char s1[N+1], s2[N+1];

  // Copia le stringhe costanti negli array di char
  strncpy(s1,"M'illumino\n",N);
  strncpy(s2,"d'immenso.",N);

  // Aggiunge s2 ad s1
  strncat(s1,s2,N);

  printf("%s\n",s1);

  return 0;
}
```

La funzione di concatenzione `strncat()` *aggiunge* la stringa `s2` a `s1`. La prima occorrenza del carattere nullo in `s1` viene sovrascritta dal primo carattere (non nullo) di `s2`. Viene aggiunto un carattere nullo ad `s1` dopo l'ultimo carattere ricopiato di `s2`.

Funzioni nella libreria string: esempi 3/3

- Possibile implementazione della funzione strcmp().

```
1 int mystrcmp(const char s1[], const char s2[]) {
2   int i=0;
3   while(s1[i] && s2[i] && s1[i]==s2[i]) i++;
4   return s1[i]-s2[i];
5 }
```

A riga 4: s1[i] $\neq$ s2[i], oppure s1[i] = s2[i] = '\0'.

Funzioni nella libreria string: esempi 3/3

- Possibile implementazione della funzione strcmp().

```
1 int mystrcmp(const char s1[], const char s2[]) {
2   int i=0;
3   while(s1[i] && s2[i] && s1[i]==s2[i]) i++;
4   return s1[i]-s2[i];
5 }
```

A riga 4: s1[i] $\neq$ s2[i], oppure s1[i] = s2[i] = '\0'.

- Possibile implementazione della funzione strncmp().

```
1 int mystrncmp(const char s1[], const char s2[], size_t n) {
2   int i=0;
3   while(i<n && s1[i] && s2[i] && s1[i]==s2[i]) i++;
4   return s1[i]-s2[i];
5 }
```

A riga 4: s1[i] $\neq$ s2[i], oppure s1[i] = s2[i] = '\0' con $0 \leq i < n$.

Funzioni nella libreria string: esempi 3/3

- Possibile implementazione della funzione strcmp().

```
1 int mystrcmp(const char s1[], const char s2[]) {
2   int i=0;
3   while(s1[i] && s2[i] && s1[i]==s2[i]) i++;
4   return s1[i]-s2[i];
5 }
```

A riga 4: s1[i] $\neq$ s2[i], oppure s1[i] = s2[i] = '\0'.

- Possibile implementazione della funzione strncmp().

```
1 int mystrncmp(const char s1[], const char s2[], size_t n) {
2   int i=0;
3   while(i<n && s1[i] && s2[i] && s1[i]==s2[i]) i++;
4   return s1[i]-s2[i];
5 }
```

A riga 4: s1[i] $\neq$ s2[i], oppure s1[i] = s2[i] = '\0' con $0 \leq i < n$.

- Possibile implementazione della funzione strlen().

```
1 size_t strlen(const char s[]) {
2   size_t i = 0;
3   while(s[i]) i++;
4   return i;
5 }
```

Nota. *Ricordiamo che size_t è un tipo intero senza segno la cui dimensione dipende dall'implementazione.*

I/O di stringhe

- Le funzioni di I/O specifiche per stringhe sono contenute nella libreria `stdio`.

Classe	Prototipo
Funzioni di input	`char *gets(char *s);` `char *fgets(char *s, int n, FILE *stream);`
Funzioni di output	`int puts(const char *s);` `int fputs(const char *s, FILE *stream);`
Funzioni di conversione	`int sscanf(const char *restrict s, const char *restrict format, ...);`

- Le funzioni per leggere stringhe ritornano l'indirizzo del primo byte della stringa, oppure `NULL` se ci sono errori di lettura.
- Le funzioni per stampare stringhe ritornano 0 se non ci sono problemi e `EOF` in caso di errore.
- La funzione `sscanf()` è molto comoda per *parsare* il contenuto di una stringa. Usa lo stesso formato della `scanf()`, ma legge l'input dalla stringa `s` non da tastiera.
- La funzione `gets()` è non sicura e assolutamente da evitare: perché?
- Possiamo leggere stringhe anche utilizzando la funzione `scanf()`: utilizziamo lo specificatore di formato `%s`. Anche in questo caso, l'utilizzo è ampiamente deprecato.

I/O di stringhe: buffer overflow 1/3

```c
#include  <stdio.h>

#define MAX 3 // Numero massimo di tentativi

// Legge username e password. Ritorna 1 se l'accesso è
// garantito, 0 altrimenti.
int check_access();

int main() {
  int i=0,check;

  do {
    check = check_access();
  } while(++i<MAX && !check);

  if(!check) {
    printf("** Access denied **\n");
    return 1;
  } else printf("** Welcome, Administrator **\n");
  // Parte l'interfaccia per l'amministratore
  return 0;
}
```

- Esempio semplificato di interfaccia per un sistema protetto da password di accesso.
- Il sistema utilizza una funzione di libreria `check_access()` che legge *username*, *password* e verifica che le credenziali immesse siano valide per l'accesso al sistema.

I/O di stringhe: buffer overflow 2/3

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#define N 16 // Max 15 caratteri per password crittate
#define M 7  // Max 6 caratteri per password in chiaro

#define ROWS(x) (sizeof(x)/sizeof(x[0])) // Calcola numero di righe in array

static char salt[] = "$1$abcdefgh$";      // Codice crittografico

// Coppie username-password (crittate). Normalmente sono in un file esterno.
static const char users[][2][N] = {
  {{"admin1"},{"$1MbDqeVITVmo"}},
  {{"admin2"},{"$1nqJS8DoIhFA"}},
  {{"admin3"},{"$1d2n7Q0.r54s"}}
};

int check_access() {
 int  check = 0, i;
 char passwd[M], usrnm[M];

  printf("Username: ");
  gets(usrnm);           // Legge username. WARNING: gets()!!!
  printf("Password: ");
  //system("stty -echo"); // Disabilita echo (Unix)
  gets(passwd);          // Legge password (in chiaro). WARNING: gets()!!!
  //system("stty echo");  // Ripristina echo (Unix)
  printf("\n\n");

  // Verifica uno ad uno tutti gli account. cript: funzione crittografica di libreria (Unix)
  for(i=ROWS(users)-1; i>=0 && !check; i--)
    check = (strcmp(users[i][0],usrnm)==0) && (strcmp(users[i][1],crypt(passwd,salt))==0);
  // Ritorna 1 "solo" se username e password sono nella lista, altrimenti 0
  return check;
}
```

I/O di stringhe: buffer overflow 3/3

- Cerchiamo di capire perché il codice presentato nelle slide precedenti è insicuro.
- Notiamo innanzitutto che la variabile `check` e i buffer di lettura `passwd[]` , `usrnm[]`, sono locali alla funzione `check_access()`: condividono lo stesso record di attivazione al momento della chiamata.
  - E' molto probabile che operazioni di overflow sui due buffer di lettura possano sovrascrivere il contenuto della variabile `check`.
  - Attenzione: l'effettiva disposizione sul record di attivazione delle variabili `check`, `passwd[]` e `usrnm[]` dipende dal compilatore e dal calcolatore.
- Notiamo che il codice fa uso della `gets()` per leggere input da tastiera. La `gets()` non permette di limitare il numero di caratteri letti.
  - Una sequenza di caratteri sufficientemente lunga (immessa da tastiera) manda in overflow i due buffer di lettura, che possono contenere solo 7 caratteri.
  - Aumentare la dimensione dei buffer di lettura non risolve il problema.
- Notiamo infine che nel main del programma, l'unico controllo che viene effettuato sul valore ritornato dalla funzione `check_access()` è che sia diverso da zero.
  - E' alquanto semplice modificare il valore della variabile `check` in modo che diventi diverso da 0, mandando in overflow uno dei due buffer di lettura.
  - Con pochi tentativi, riusciamo ad ottenere l'accesso al sistema senza conoscere né password né username.

Morris worm

- Il Morris worm è uno dei primi esempi di **worm** diffuso via Internet nel 1988
- Storicamente, il Morris worm è il **primo caso** di attacco di tipo informatico
  - ad aver ottenuto attenzione mediatica,
  - punito con una condanna per pirateria informatica negli Stati Uniti.
- Definizioni:
  - **Virus**. Frammento di software che, una volta eseguito, è in grado di infettare altri file in modo da auto-replicarsi.
  - **Worm**. Software in grado di auto-replicarsi diffondendosi su altre macchine tramite connessioni di rete. A differenza dei virus, non necessita di infettare altri file per diffondersi.

Morris worm

- Il Morris worm è uno dei primi esempi di **worm** diffuso via Internet nel 1988
- Storicamente, il Morris worm è il **primo caso** di attacco di tipo informatico
  - ad aver ottenuto attenzione mediatica,
  - punito con una condanna per pirateria informatica negli Stati Uniti.
- Definizioni:
  - **Virus**. Frammento di software che, una volta eseguito, è in grado di infettare altri file in modo da auto-replicarsi.
  - **Worm**. Software in grado di auto-replicarsi diffondendosi su altre macchine tramite connessioni di rete. A differenza dei virus, non necessita di infettare altri file per diffondersi.
- Secondo il suo creatore Morris, la diffusione del worm non doveva avere lo scopo di causare danni ma di valutare le dimensione di Internet.
  - Nel 1988 la rete Internet collegava circa 60.000 calcolatori.
- In pratica, il Morris worm provocò lo stallo di numerose macchine connesse alla rete, causando diverse migliaia di dollari di danni economici.

Morris worm: i fatti

- Il 2 Novembre 1988, Robert Morris, allora studente presso la Cornell University (New York), manda in esecuzione il worm su una macchina del MIT (Boston).
- Prima di entrare in un computer il worm verifica che non sia già presente una copia.
  - Se una copia non è presente, il worm utilizza alcune falle di sicurezza dei sistemi Unix per installarsi sulla macchina. Una volta entrato nella macchina, individua in alcuni file di sistema gli indirizzi di altri calcolatori sulla rete e ripete il processo di replicazione su queste macchine.
  - Se il worm è già presente sulla macchina, questa viene ignorata. Per aggirare alcuni semplici sistemi di protezione da parte di amministratori *accorti*, con probabilità 1 su 7, il worm prova in ogni caso ad infettare la macchina, anche se su questa è già presente.
- Il livello di replicazione risulta essere eccessivo: il worm si diffonde rapidamente, infettando alcune macchine più volte e rendendole in poco tempo inutilizzabili.
- Nel giro di 24 ore, circa il 10% dei calcolatori connessi alla rete sono infetti. Tra questi ci sono computer appartenenti alla NASA, Pentagono, MIT, università di Berkeley e Stanford.
- Il danno economico viene stimato tra in 100.000 e i 10.000.000 di dollari.
- Il 22 Gennaio 1990, Robert Morris viene condannato a 3 anni di libertà condizionata, una multa di 10.000 dollari e 400 ore di servizi sociali.
- Attualmente, Rober Morris insegna al MIT.

Morris worm: i punti di attacco

- Il Morris worm sfrutta alcune debolezze dei sistemi Unix nel 1988 (adesso risolte) e ingenuità da parte degli utenti (non risolte).
- I dettagli sul funzionamento del worm non sono banali. Esiste un report molto completo e dettagliato (datato 1988).

Morris worm: i punti di attacco

- Il Morris worm sfrutta alcune debolezze dei sistemi Unix nel 1988 (adesso risolte) e ingenuità da parte degli utenti (non risolte).
- I dettagli sul funzionamento del worm non sono banali. Esiste un report molto completo e dettagliato (datato 1988).
- Elenchiamo unicamente le caratteristiche principali, senza addentrarci nei dettagli:
  - **Utility finger**. E' una utility dei sistemi Unix che permette di ottenere informazioni sugli utenti attualmente connessi alla macchina (anche in remoto). L'implementazione del *demone* **fingerd** in ascolto sulla macchina faceva uso della `gets()` per leggere le richieste in input. Il Morris worm sfruttava questa debolezza per mandare in overflow il buffer di lettura del demone.

Morris worm: i punti di attacco

- Il Morris worm sfrutta alcune debolezze dei sistemi Unix nel 1988 (adesso risolte) e ingenuità da parte degli utenti (non risolte).
- I dettagli sul funzionamento del worm non sono banali. Esiste un report molto completo e dettagliato (datato 1988).
- Elenchiamo unicamente le caratteristiche principali, senza addentrarci nei dettagli:
  - **Utility finger**. E' una utility dei sistemi Unix che permette di ottenere informazioni sugli utenti attualmente connessi alla macchina (anche in remoto). L'implementazione del *demone* **fingerd** in ascolto sulla macchina faceva uso della `gets()` per leggere le richieste in input. Il Morris worm sfruttava questa debolezza per mandare in overflow il buffer di lettura del demone.
  - Utility **sendmail**. E' una utility dei sistemi Unix per gestire la posta elettronica. Il Morris worm sfruttava una funzionalità di debugging di *sendmail* per eseguire dei comandi sulla macchina ospite.

Morris worm: i punti di attacco

- Il Morris worm sfrutta alcune debolezze dei sistemi Unix nel 1988 (adesso risolte) e ingenuità da parte degli utenti (non risolte).
- I dettagli sul funzionamento del worm non sono banali. Esiste un report molto completo e dettagliato (datato 1988).
- Elenchiamo unicamente le caratteristiche principali, senza addentrarci nei dettagli:
  - **Utility finger**. E' una utility dei sistemi Unix che permette di ottenere informazioni sugli utenti attualmente connessi alla macchina (anche in remoto). L'implementazione del *demone* **fingerd** in ascolto sulla macchina faceva uso della `gets()` per leggere le richieste in input. Il Morris worm sfruttava questa debolezza per mandare in overflow il buffer di lettura del demone.
  - Utility **sendmail**. E' una utility dei sistemi Unix per gestire la posta elettronica. Il Morris worm sfruttava una funzionalità di debugging di *sendmail* per eseguire dei comandi sulla macchina ospite.
  - **Password deboli**. Il Morris worm include codice per decrittare password banali di utenti del sistema (riuscendoci con buona probabilità).

Morris worm: i punti di attacco

- Il Morris worm sfrutta alcune debolezze dei sistemi Unix nel 1988 (adesso risolte) e ingenuità da parte degli utenti (non risolte).
- I dettagli sul funzionamento del worm non sono banali. Esiste un report molto completo e dettagliato (datato 1988).
- Elenchiamo unicamente le caratteristiche principali, senza addentrarci nei dettagli:
  - **Utility finger**. E' una utility dei sistemi Unix che permette di ottenere informazioni sugli utenti attualmente connessi alla macchina (anche in remoto). L'implementazione del *demone* **fingerd** in ascolto sulla macchina faceva uso della `gets()` per leggere le richieste in input. Il Morris worm sfruttava questa debolezza per mandare in overflow il buffer di lettura del demone.
  - Utility **sendmail**. E' una utility dei sistemi Unix per gestire la posta elettronica. Il Morris worm sfruttava una funzionalità di debugging di *sendmail* per eseguire dei comandi sulla macchina ospite.
  - **Password deboli**. Il Morris worm include codice per decrittare password banali di utenti del sistema (riuscendoci con buona probabilità).
- Alcuni punti di attacco sfruttati dal Morris worm sono tutt'ora *validi*:
  - utenti che utilizzano password banali,
  - software che faccia uso di `gets()` (deprecata ma ancora utilizzabile), o anche altre funzioni di libreria che non permettono di controllare la lunghezza del buffer su cui si effettuano operazioni di scrittura.