Password File

- ASCII text: /etc/passwd
- Passed structure in ¡pwd.h¿
- **root**: superuser, UID=0
  root:x:0:0:root:/root:/bin/bash
- **guest**: no privileges
  nobody:x:65534:35534:Nobody:/home/bin/sh
- normal account
  elglaly:x:115:125:Yasmine Elg:/home/elglaly:/bin/bash
- Encrypted passwords /etc/shadow or /etc/master.passwd
- Shadow is readable only by root
- /etc/passwd is world-readable
- Cannot access encrypted passwords!

Group File

- /etc/group
  - wheel (BSD)
  - gid 0

Description	struct group member
group name	char *gr_name
encrypted password	char *gr_passwd
group ID	int gr_gid
array to names	char **gr_mem

Group Info

```
#include <grp.h>
struct group *getgrgid(gid_t gid);
struct group *getgrnam(const char *name);
        Return: pointer if OK, NULL on error
struct group *getgrent(void); // read next entry
        Returns: pointer if OK, NULL on error/EOF
void setgrent(void); //rewind group file
void endgrent(void); // close group file
```

Supplementary

```
#include <unistd.h>
int getgroups(int gidsetsize, gid_t grouplist[]);
    //Fills grouplist with gidsetsize group IDs
        Returns: #supp GIDs if OK, -1 on error

#include <grp.h> //on Linux
#include <unistd.h> //on FreeBSD, Mac OS X, Solaris
int setgroups(int ngroups, const gid_t grouplist[]);
    //set supplementary GID for calling process
int initgroups(const char *username, gid_t basegid);
//reads entire group file (getgrent, setgrent, endgrent)
        Both return: 0 if OK, -1 on error


struct passwd { /* Linux version */
    char *pw_name;  /* username */
    char *pw_passwd; /* encrypted password */
    uid_t pw_uid;   /* user ID */
    gid_t pw_gid;   /* group ID */
    char *pw_gecos; /* general info */
    char *pw_dir;   /* home directory */
    char *pw_shell; /* shell program */
};
```

Memory Allocation - Space Calculations

sizeof for basic types

sizeof (char)	= 1
sizeof (short)	= 2
sizeof (int)	= 4
sizeof (float)	= 4
sizeof (long)	= 8
sizeof (double)	= 8
sizeof (char *)	= 4 / 8

sizeof for array types

```
double sample[100];
sizeof(sample) = 100 * 8 = 800
char string[81];
sizeof(string) = 81 * 1 = 81
```

BUT

```
void foo(char buffer[81]) { . . . }
sizeof(buffer); // = 4 or 8 !!
```

Array arguments are really pointers!

```
struct tm {
    int tm_sec; /* Seconds (0-60) */
    int tm_min; /* Minutes (0-59) */
    int tm_hour; /* Hours (0-23) */
    int tm_mday; /* Day of the month (1-31) */
    int tm_mon; /* Month (0-11) */
    int tm_year; /* Year - 1900 */
    int tm_wday; /* Day of the week (0-6, Sunday = 0) */
    int tm_yday; /* Day in the year (0-365, 1 Jan = 0) */
    int tm_isdst; /* Daylight saving time */
};
```

Fields in /etc/shadow

Description	struct passwd member
username	char *pw_name
encrypted password	char *pw_passwd
user ID	uid_t pw_uid
group ID	gid_t pw_gid
comment	char *pw_gecos
workding directory	char *pw_dir
shell program	char *pw_shell
access class	char *pw_class
time to change passwd	time_t pw_change
account expiration date	time_t pw_expire

Fields in /etc/passwd

Description	struct spwd member
user login name	char *sp_namp
encrypted password	char *sp_pwdp
date of last change	int sp_lstchg
days until change allowed	int sp_min
days until change required	int sp_max
days before warning	int sp_warn
days before account inactive	int sp_inact
date when account expires	int sp_expire
reserved	unsigned int sp_flag

Program Access

Fetching Entries

```
#include <pwd.h>
struct passwd *getpwuid(uid_t uid); // used by ls
struct passwd *getpwnam(const char *name);
            // used by login
        Return: pointer to passwd struct if OK,
                NULL on error
```

Iteration

```
struct passwd *getpwent(void);
            // Opens necessary files
        Return: pointer to passwd struct if OK,
                NULL on error/EOF
void setpwent(void); // rewinds files
void endpwent(void); // closes files
```

Shadow Passwords

```
#include <shadow.h>
struct spwd *getspnam(const char *name);
struct spwd *getspent(void);
        Return: pointer if OK, NULL on error
void setspent(void);
void endspent(void);
```

Dynamic Memory Allocation

```
#include <stdlib.h>
void *malloc(size_t size);
void *calloc(size_t nobj, size_t size);
void *realloc(void *ptr, size_t newsize);
    Returns: pointer on success, NULL otherwise
void free(void *ptr);
```

void *malloc(unsigned nbytes)

- Allocates nbytes of memory
- Guaranteed not to overlap other allocated memory
- Returns point to first byte (or NULL if heap is full)
- Similar to constructor in Java - allocates space
- Allocated space is uninitialized (random garbage)

void free(void *ptr)

- Frees the memory assigned to ptr.
- The space must have been allocated by malloc
- No garbage collection in C
- Can slowly consume memory if not careful

Time

```
#include <time.h>
time_t time(time_t *calptr);
       Returns: value of time if OK, -1 on error
Number of seconds since Epoch: 00:00:00 1970/1/1, UTC
Example: curtime = time (NULL); /* Get the current time. */

#include <sys/time.h>
int gettimeofday(struct timeval *restrict tp,
    void *restrict tzp);
       Returns: 0 always
struct timeval {
    time_t tv_sec; /* sec */
    long tv_usec; /*microsec*/
};
```