

Libreria «stdbool.h» .....	1911
Libreria «time.h» .....	1912
Libreria «ctype.h» .....	1912
Libreria «stdint.h» .....	1913
Libreria «inttypes.h» .....	1914
Libreria «stdarg.h» .....	1917
Libreria «stddef.h» .....	1917
Libreria «stdlib.h» .....	1917
Libreria «string.h» .....	1918
Libreria «stdio.h» .....	1920

atoi.c 1917 ctype.h 1912 inttypes.h 1914 memcpy.c 1918 memset.c 1918 NULL.h 1911 ptrdiff\_t.h 1911 restrict.h 1911 size\_t.h 1911 snprintf.c 1920 stdarg.h 1917 stdbool.h 1911 stddef.h 1917 stdint.h 1913 stdio.h 1920 stdlib.h 1917 string.h 1918 strncpy.c 1918 time.h 1912 vsnprintf.c 1920 wchar\_t.h 1911

Secondo lo standard, più file di libreria dichiarano gli stessi tipi speciali e le stesse costanti. Per evitare confusione, la dichiarazione di queste costanti e di questi tipi condivisi, viene collocata in file isolati che, successivamente, altri file incorporano a seconda della necessità. Inoltre, il compilatore usato per la costruzione di questo sistema non gestisce i «puntatori ristretti», ovvero non considera valida la parola chiave **restrict**. Per mantenere una forma aderente allo standard si aggiunge la dichiarazione della macro-variabile **restrict** vuota, in un file separato che molti altri file incorporano.

Listato u166.1. './05/include/restrict.h'

```
#ifndef _RESTRICT_H
#define _RESTRICT_H    1

#define restrict

#endif
```

Listato u166.2. './05/include/NULL.h'

```
#ifndef _NULL_H
#define _NULL_H        1

#define NULL 0

#endif
```

Listato u166.3. './05/include/ptrdiff\_t.h'

```
#ifndef _PTRDIFF_T_H
#define _PTRDIFF_T_H   1

typedef long int ptrdiff_t;

#endif
```

Listato u166.4. './05/include/size\_t.h'

```
#ifndef _SIZE_T_H
#define _SIZE_T_H      1

typedef unsigned long int size_t;

#endif
```

Listato u166.5. './05/include/wchar\_t.h'

```
#ifndef _WCHAR_T_H
#define _WCHAR_T_H    1

typedef unsigned char wchar_t;

#endif
```

Dal file 'wchar\_t.h' si comprende che, per il sistema in corso di realizzazione, si intende gestire al massimo la codifica ASCII e nulla di più.

## Libreria «stdbool.h»

«

Listato u166.6. './05/include/stdbool.h'

```
#ifndef _STDBOOL_H
#define _STDBOOL_H    1

#define bool    _Bool
#define true    1
#define false   0
#define __bool_true_false_are_defined  1

#endif
```

## Libreria «time.h»

«

Listato u166.7. './05/include/time.h'

```
#ifndef _TIME_H
#define _TIME_H    1

#include <restrict.h>
#include <size_t.h>
#include <NULL.h>

#define CLOCKS_PER_SEC 100L
typedef long int clock_t;
typedef long int time_t;

struct tm {int tm_sec; int tm_min; int tm_hour;
           int tm_mday; int tm_mon; int tm_year;
           int tm_wday; int tm_yday; int tm_isdst;};

clock_t    clock    (void);
time_t     time     (time_t *timer);
double     difftime (time_t time1, time_t time0);
time_t     mktime   (struct tm *timeptr);
struct tm *gmtime   (const time_t *timer);
struct tm *localtime(const time_t *timer);
char *     asctime  (const struct tm *timeptr);
char *     ctime    (const time_t *timer);
size_t     strftime (char * restrict s, size_t maxsize,
                    const char * restrict format,
                    const struct tm * restrict timeptr);

#define ctime(t) (asctime (localtime (t)));

#endif
```

Del file 'time.h' viene usato solo il tipo 'clock\_t' e la macro-variabile **CLOCKS\_PER\_SEC**, con la quale si dichiara implicitamente la frequenza con cui deve reagire il temporizzatore interno del realizzando sistema. Pertanto, le funzioni del file di cui si vedono i prototipi, non vengono realizzate.

## Libreria «ctype.h»

«

Listato u166.8. './05/include/ctype.h'

```
#ifndef _CTYPE_H
#define _CTYPE_H    1

#include <NULL.h>

#define isblank(C) ((int) (C == ' ' || C == '\t'))
#define isspace(C) ((int) (C == ' ' \
                          || C == '\f' \
                          || C == '\n' \
                          || C == '\r' \
                          || C == '\t' \
                          || C == '\v'))
#define isdigit(C) ((int) (C >= '0' && C <= '9'))
#define isxdigit(C) ((int) ((C >= '0' && C <= '9' \
                             || (C >= 'A' && C <= 'F') \
                             || (C >= 'a' && C <= 'f'))))
#define isupper(C) ((int) (C >= 'A' && C <= 'Z'))
#define islower(C) ((int) (C >= 'a' && C <= 'z'))
#define iscntrl(C) \
    ((int) ((C >= 0x00 && C <= 0x1F) || C == 0x7F))
#define isgraph(C) ((int) (C >= 0x21 && C <= 0x7E))
#define isprint(C) ((int) (C >= 0x20 && C <= 0x7E))
#define isalpha(C) (isupper (C) || islower (C))
```

1912

```
#define isalnum(C) (isalpha (C) || isdigit (C))
#define ispunct(C) \
    (isgraph (C) && (!isspace (C)) && (!isalnum (C)))

#endif
```

## Libreria «stdint.h»

«

Listato u166.9. './05/include/stdint.h'

```
#ifndef _STDINT_H
#define _STDINT_H    1

typedef signed char    int8_t;
typedef short int     int16_t;
typedef int            int32_t;    // x86-32
typedef unsigned char  uint8_t;
typedef unsigned short int  uint16_t;
typedef unsigned int   uint32_t;    // x86-32

#define INT8_MIN        (-0x80)
#define INT8_MAX        (0x7F)
#define UINT8_MAX       (0xFF)
#define INT16_MIN       (-0x8000)
#define INT16_MAX       (0x7FFF)
#define UINT16_MAX      (0xFFFF)
#define INT32_MIN       (-0x80000000)
#define INT32_MAX       (0x7FFFFFFF)
#define UINT32_MAX      (0xFFFFFFFF)

typedef signed char    int_least8_t;
typedef short int     int_least16_t;
typedef int            int_least32_t;
typedef unsigned char  uint_least8_t;
typedef unsigned short int  uint_least16_t;
typedef unsigned int   uint_least32_t;

#define INT_LEAST8_MIN    (-0x80)
#define INT_LEAST8_MAX    (0x7F)
#define UINT_LEAST8_MAX   (0xFF)
#define INT_LEAST16_MIN   (-0x8000)
#define INT_LEAST16_MAX   (0x7FFF)
#define UINT_LEAST16_MAX  (0xFFFF)
#define INT_LEAST32_MIN   (-0x80000000)
#define INT_LEAST32_MAX   (0x7FFFFFFF)
#define UINT_LEAST32_MAX  (0xFFFFFFFF)

#define INT8_C(VAL)      VAL
#define INT16_C(VAL)     VAL
#define INT32_C(VAL)     VAL
#define UINT8_C(VAL)     VAL
#define UINT16_C(VAL)    VAL
#define UINT32_C(VAL)    VAL ## U

typedef signed char    int_fast8_t;
typedef int            int_fast16_t;
typedef int            int_fast32_t;
typedef unsigned char  uint_fast8_t;
typedef unsigned int   uint_fast16_t;
typedef unsigned int   uint_fast32_t;

#define INT_FAST8_MIN    (-0x80)
#define INT_FAST8_MAX    (0x7F)
#define UINT_FAST8_MAX   (0xFF)
#define INT_FAST16_MIN   (-0x80000000)
#define INT_FAST16_MAX   (0x7FFFFFFF)
#define UINT_FAST16_MAX  (0xFFFFFFFF)
#define INT_FAST32_MIN   (-0x80000000)
#define INT_FAST32_MAX   (0x7FFFFFFF)
#define UINT_FAST32_MAX  (0xFFFFFFFF)

typedef int            intptr_t;
typedef unsigned int   uintptr_t;

#define INTPTR_MIN       (-0x80000000)
#define INTPTR_MAX       (0x7FFFFFFF)
#define UINTPTR_MAX      (0xFFFFFFFF)

typedef long int       intmax_t;
typedef unsigned long int  uintmax_t;

#define INTMAX_C(VAL)    VAL ## L
```

1913

```

#define UINTMAX_C(VAL) VAL ## UL

#define INTMAX_MIN      (-0x80000000L)
#define INTMAX_MAX      (0x7FFFFFFFL)
#define UINTMAX_MAX     (0xFFFFFFFFFUL)

#define PTRDIFF_MIN     (-0x80000000)
#define PTRDIFF_MAX     (0x7FFFFFFF)

#define SIG_ATOMIC_MIN  (-0x80000000)
#define SIG_ATOMIC_MAX  (0x7FFFFFFF)

#define SIZE_MAX        (0xFFFFFFFFFU)

#define WCHAR_MIN       (0)
#define WCHAR_MAX       (0xFFFFFU)

#define WINT_MIN        (-0x8000L)
#define WINT_MAX        (0x7FFFL)

#endif

```

## Libreria <inttypes.h>

<

Listato ul66.10. './05/include/inttypes.h'

```

#ifndef _INTTYPES_H
#define _INTTYPES_H 1

#include <restrict.h>
#include <stdint.h>
#include <wchar_t.h>

typedef struct {intmax_t quot; intmax_t rem;} imaxdiv_t;

#define PRId8          "d"
#define PRId16         "d"
#define PRId32         "d"
#define PRId64         "lld"
#define PRIdLEAST8    "d"
#define PRIdLEAST16   "d"
#define PRIdLEAST32   "d"
#define PRIdLEAST64   "lld"
#define PRIdFAST8     "d"
#define PRIdFAST16    "d"
#define PRIdFAST32    "d"
#define PRIdFAST64    "lld"
#define PRIdMAX       "lld"
#define PRIdPTR       "d"
#define PRIi8         "i"
#define PRIi16        "i"
#define PRIi32        "i"
#define PRIi64        "lli"
#define PRIiLEAST8   "i"
#define PRIiLEAST16  "i"
#define PRIiLEAST32  "i"
#define PRIiLEAST64  "lli"
#define PRIiFAST8    "i"
#define PRIiFAST16   "i"
#define PRIiFAST32   "i"
#define PRIiFAST64   "lli"
#define PRIiMAX      "lli"
#define PRIiPTR      "i"
#define PRIB8        "b" // PRIB... non è standard
#define PRIB16       "b" //
#define PRIB32       "b" //
#define PRIB64       "llb" //
#define PRIBLEAST8   "b" //
#define PRIBLEAST16  "b" //
#define PRIBLEAST32  "b" //
#define PRIBLEAST64  "llb" //
#define PRIBFAST8    "b" //
#define PRIBFAST16   "b" //
#define PRIBFAST32   "b" //
#define PRIBFAST64   "llb" //
#define PRIBMAX      "llb" //
#define PRIBPTR      "b" //
#define PRIO8        "o"
#define PRIO16       "o"
#define PRIO32       "o"
#define PRIO64       "llo"

```

```

#define PRIOLEAST8    "o"
#define PRIOLEAST16  "o"
#define PRIOLEAST32  "o"
#define PRIOLEAST64  "llo"
#define PRIOFAST8    "o"
#define PRIOFAST16   "o"
#define PRIOFAST32   "o"
#define PRIOFAST64   "llo"
#define PRIOMAX      "llo"
#define PRIOPTR      "o"
#define PRIu8        "u"
#define PRIu16       "u"
#define PRIu32       "u"
#define PRIu64       "llu"
#define PRIULEAST8   "u"
#define PRIULEAST16  "u"
#define PRIULEAST32  "u"
#define PRIULEAST64  "llu"
#define PRIUFAST8    "u"
#define PRIUFAST16   "u"
#define PRIUFAST32   "u"
#define PRIUFAST64   "llu"
#define PRIUMAX      "llu"
#define PRIUPTR      "u"
#define PRIx8        "x"
#define PRIx16       "x"
#define PRIx32       "x"
#define PRIx64       "llx"
#define PRIXLEAST8   "x"
#define PRIXLEAST16  "x"
#define PRIXLEAST32  "x"
#define PRIXLEAST64  "llx"
#define PRIXFAST8    "x"
#define PRIXFAST16   "x"
#define PRIXFAST32   "x"
#define PRIXFAST64   "llx"
#define PRIXMAX      "llx"
#define PRIXPTR      "x"
#define PRIx8        "X"
#define PRIx16       "X"
#define PRIx32       "X"
#define PRIx64       "llX"
#define PRIXLEAST8   "X"
#define PRIXLEAST16  "X"
#define PRIXLEAST32  "X"
#define PRIXLEAST64  "llX"
#define PRIXFAST8    "X"
#define PRIXFAST16   "X"
#define PRIXFAST32   "X"
#define PRIXFAST64   "llX"
#define PRIXMAX      "llX"
#define PRIXPTR      "X"

#define SCNd8        "hhd"
#define SCNd16       "hd"
#define SCNd32       "d"
#define SCNd64       "lld"
#define SCNGLEAST8   "hhd"
#define SCNGLEAST16  "hd"
#define SCNGLEAST32  "d"
#define SCNGLEAST64  "lld"
#define SCNGFAST8    "hhd"
#define SCNGFAST16   "d"
#define SCNGFAST32   "d"
#define SCNGFAST64   "lld"
#define SCNGMAX      "lld"
#define SCNGPTR      "d"
#define SCNi8        "hhi"
#define SCNi16       "hi"
#define SCNi32       "i"
#define SCNi64       "lli"
#define SCNiLEAST8   "hhi"
#define SCNiLEAST16  "hi"
#define SCNiLEAST32  "i"
#define SCNiLEAST64  "lli"
#define SCNiFAST8    "hhi"
#define SCNiFAST16   "i"
#define SCNiFAST32   "i"
#define SCNiFAST64   "lli"
#define SCNiMAX      "lli"
#define SCNiPTR      "i"

```

```

#define SCNb8 "hhb" // SCNb... non è standard
#define SCNb16 "hb" //
#define SCNb32 "b" //
#define SCNb64 "llb" //
#define SCNbLEAST8 "hhb" //
#define SCNbLEAST16 "hb" //
#define SCNbLEAST32 "b" //
#define SCNbLEAST64 "llb" //
#define SCNbFAST8 "hhb" //
#define SCNbFAST16 "b" //
#define SCNbFAST32 "b" //
#define SCNbFAST64 "llb" //
#define SCNbMAX "llb" //
#define SCNbPTR "b" //
#define SCNo8 "hho" //
#define SCNo16 "ho" //
#define SCNo32 "o" //
#define SCNo64 "llo" //
#define SCNoLEAST8 "hho" //
#define SCNoLEAST16 "ho" //
#define SCNoLEAST32 "o" //
#define SCNoLEAST64 "llo" //
#define SCNoFAST8 "hho" //
#define SCNoFAST16 "o" //
#define SCNoFAST32 "o" //
#define SCNoFAST64 "llo" //
#define SCNoMAX "llo" //
#define SCNoPTR "o" //
#define SCNu8 "hhu" //
#define SCNu16 "hu" //
#define SCNu32 "u" //
#define SCNu64 "llu" //
#define SCNuLEAST8 "hhu" //
#define SCNuLEAST16 "hu" //
#define SCNuLEAST32 "u" //
#define SCNuLEAST64 "llu" //
#define SCNuFAST8 "hhu" //
#define SCNuFAST16 "u" //
#define SCNuFAST32 "u" //
#define SCNuFAST64 "llu" //
#define SCNuMAX "llu" //
#define SCNuPTR "u" //
#define SCNx8 "hhx" //
#define SCNx16 "hx" //
#define SCNx32 "x" //
#define SCNx64 "llx" //
#define SCNxLEAST8 "hhx" //
#define SCNxLEAST16 "hx" //
#define SCNxLEAST32 "x" //
#define SCNxLEAST64 "llx" //
#define SCNxFAST8 "hhx" //
#define SCNxFAST16 "x" //
#define SCNxFAST32 "x" //
#define SCNxFAST64 "llx" //
#define SCNxMAX "llx" //
#define SCNxPTR "x" //

imaxdiv_t imaxdiv (intmax_t numer, intmax_t denom);
intmax_t strtouimax (const char *restrict nptr,
char **restrict endptr,
int base);
uintmax_t strtouimax (const char *restrict nptr,
char **restrict endptr,
int base);
intmax_t wcstouimax (const wchar_t *restrict nptr,
wchar_t **restrict endptr,
int base);
uintmax_t wcstouimax (const wchar_t *restrict nptr,
wchar_t **restrict endptr,
int base);
#endif

```

La libreria 'inttypes.h' serve per le macro-variabili del tipo 'PRIx\*', in modo da utilizzare correttamente la funzione *printf()*, mentre si fa riferimento a tipi di valori numerici definiti nel file 'stdint.h'. Pertanto, le funzioni non vengono realizzate.

## Libreria «stdarg.h»

Listato ul66.11. './05/include/stdarg.h'

```

#ifndef _STDARG_H
#define _STDARG_H 1

typedef unsigned char *va_list;

#define va_start(ap, last) ((void) ((ap) = \
((va_list) &(last)) + (sizeof (last))))
#define va_end(ap) ((void) ((ap) = 0))
#define va_copy(dest, src) \
((void) ((dest) = (va_list) (src)))
#define va_arg(ap, type) \
(((ap) = (ap) + (sizeof (type))), \
*((type *) ((ap) - (sizeof (type)))))
#endif

```

## Libreria «stddef.h»

Listato ul66.12. './05/include/stddef.h'

```

#ifndef _STDDEF_H
#define _STDDEF_H 1

#include <ptrdiff_t.h>
#include <size_t.h>
#include <wchar_t.h>
#include <NULL.h>

#define offsetof(type, member) \
((size_t) &((type *)0)->member)

#endif

```

## Libreria «stdlib.h»

Listato ul66.13. './05/include/stdlib.h'

```

#ifndef _STDLIB_H
#define _STDLIB_H 1

#include <size_t.h>
#include <wchar_t.h>
#include <NULL.h>
#include <limits.h>
#include <restrict.h>

typedef struct {int quot; int rem;} div_t;
typedef struct {long int quot; long int rem;} ldiv_t;
typedef struct {long long int quot; long long int rem;} lldiv_t;

#define EXIT_FAILURE 1
#define EXIT_SUCCESS 0
#define RAND_MAX INT_MAX
#define MB_CUR_MAX ((size_t) MB_LEN_MAX)

int atoi (const char *nptr);
long int atol (const char *nptr);
long long int atoll (const char *nptr);
double atof (const char *nptr);

float strtof (const char * restrict nptr,
char ** restrict endptr);
double strtod (const char * restrict nptr,
char ** restrict endptr);
long double strtold (const char * restrict nptr,
char ** restrict endptr);
long int strtol (const char * restrict nptr,
char ** restrict endptr, int base);
long long int strtoll (const char * restrict nptr,
char ** restrict endptr, int base);
unsigned long int strtoul (const char * restrict nptr,
char ** restrict endptr, int base);
unsigned long long int strtoull (const char * restrict nptr,
char ** restrict endptr, int base);

int rand (void);
void srand (unsigned int seed);

void *malloc (size_t size);
void *realloc (void *ptr, size_t size);
void free (void *ptr);
#define calloc(nmemb, size) (malloc ((nmemb) * (size)))

int atexit (void (*func) (void));
void exit (int status);
void _Exit (int status);
void abort (void);

char *getenv (const char *name);
int system (const char *string);

```

```

void qsort (void *base,
           size_t nmemb,
           size_t size,
           int (*compar) (const void *, const void *));
void *bsearch (const void *key,
              const void *base,
              size_t nmemb,
              size_t size,
              int (*compar) (const void *, const void *));

int abs (int j);
long int labs (long int j);
long long int llabs (long long int j);

div_t div (int numer, int denom);
ldiv_t ldiv (long int numer, long int denom);
lldiv_t lldiv (long long int numer, long long int denom);

int mblen (const char *s, size_t n);
int mbtowc (wchar_t *restrict pwc, const char *restrict s, size_t n);
int wctomb (char *s, wchar_t wc);
size_t mbstowcs (wchar_t *restrict pwcs, const char *restrict s, size_t n);
size_t wctombs (char *restrict s, const wchar_t *restrict pwcs, size_t n);
#endif

```

Di questa libreria vengono realizzate solo alcune funzioni, ma in particolare, *\_Exit()*, *malloc()*, *realloc()* e *free()*, dipendono strettamente dal contesto del sistema; pertanto vengono mostrate a parte, in un'altra sezione più specifica.

Listato ul66.14. './05/lib/atoi.c'

```

#include <stdlib.h>
#include <ctype.h>
int
atoi (const char *nptr)
{
    int i;
    int sign = +1;
    int n;

    for (i = 0; isspace (nptr[i]); i++)
        ; // Si limita a saltare gli spazi iniziali.

    if (nptr[i] == '+')
        {
            sign = +1;
            i++;
        }
    else if (nptr[i] == '-')
        {
            sign = -1;
            i++;
        }

    for (n = 0; isdigit (nptr[i]); i++)
        {
            // Accumula il valore.
            n = (n * 10) + (nptr[i] - '0');
        }

    return sign * n;
}

```

## Libreria «string.h»

Listato ul66.15. './05/include/string.h'

```

#ifndef _STRING_H
#define _STRING_H    1

#include <restrict.h>
#include <size_t.h>
#include <NULL.h>

void *memcpy (void *restrict dst, const void *restrict org,
             size_t n);
void *memmove (void *dst, const void *org, size_t n);

char *strcpy (char *restrict dst,
             const char *restrict org);
char *strncpy (char *restrict dst, const char *restrict org,
             size_t n);

```

1918

```

char *strcat (char *restrict dst, const char *restrict org);
char *strncat (char *restrict dst, const char *restrict org,
             size_t n);

int memcmp (const void *s1, const void *s2, size_t n);
int strcmp (const char *s1, const char *s2);
int strcoll (const char *s1, const char *s2);
int strncmp (const char *s1, const char *s2, size_t n);
size_t strxfrm (char *restrict dst,
               const char *restrict org, size_t n);

void *memchr (const void *s, int c, size_t n);
char *strchr (const char *s, int c);
char *strrchr (const char *s, int c);
size_t strspn (const char *s, const char *accept);
size_t strcspn (const char *s, const char *reject);
char *strpbrk (const char *s, const char *accept);
char *strstr (const char *string, const char *substring);
char *strtok (char *restrict string,
             const char *restrict delim);

void *memset (void *s, int c, size_t n);
char *strerror (int errnum);
size_t strlen (const char *s);

#endif

```

Delle funzioni dichiarate nel file 'string.h' vengono realizzate solo quelle dei listati successivi.

Listato ul66.16. './05/lib/memset.c'

```

#include <string.h>
void
memset (void *s, int c, size_t n)
{
    unsigned char *a = (unsigned char *) s;
    unsigned char x = (unsigned char) c;
    size_t i;
    for (i = 0; n > 0 && i < n; i++)
        {
            a[i] = x;
        }
    return s;
}

```

Listato ul66.17. './05/lib/strncpy.c'

```

#include <string.h>
char
strncpy (char *restrict dst, const char *restrict org,
        size_t n)
{
    size_t i;
    for (i = 0; n > 0 && i < n && org[i] != 0; i++)
        {
            dst[i] = org[i];
        }
    for ( ; n > 0 && i < n; i++)
        {
            dst[i] = 0;
        }
    return dst;
}

```

Listato ul66.18. './05/lib/memcpy.c'

```

#include <string.h>
void *
memcpy (void *restrict dst, const void *restrict org,
        size_t n)
{
    unsigned char *d = (unsigned char *) dst;
    unsigned char *o = (unsigned char *) org;
    size_t i;
    for (i = 0; n > 0 && i < n; i++)
        {
            d[i] = o[i];
        }
    return dst;
}

```

1919

## Libreria «stdio.h»

<

La libreria che è rappresentata dal file «stdio.h» è la più noiosa di questo gruppo iniziale. Qui viene mostrato un file incompleto, contenente solo ciò che serve al sistema in corso di realizzazione.

Listato u166.19. «./05/include/stdio.h»

```
#ifndef _STDIO_H
#define _STDIO_H    1

#include <restrict.h>
#include <size_t.h>
#include <stdarg.h>
#include <stdint.h>
#include <kernel/vga.h>

int vsnprintf(char *restrict s, size_t n,
              const char *restrict format, va_list arg);
int snprintf(char *restrict s, size_t n,
             const char *restrict format, ...);

#define vsnprintf(s, n, format, arg) (vsnprintf(s, n, format, arg))
#define vsprintf(s, format, arg)    (vsnprintf(s, SIZE_MAX, format, arg))
#define sprintf(s, ...)             (snprintf(s, SIZE_MAX, __VA_ARGS__))

#define vprintf(format, arg)        (vga_vprintf(format, arg))
#define printf(...)                 (vga_printf(__VA_ARGS__))
#define puts(s)                     (vga_puts(s, SIZE_MAX); \
                                     vga_puts("\n", 2))
#define putchar(c)                  (vga_putc(c), c)

char *gets(char *s);

// Il resto del file «stdio.h» standard viene omissso.
#endif
```

Le uniche funzioni che si possono realizzare in modo generalizzato sono *vsnprintf()* e *snprintf()*; tuttavia, la realizzazione che viene mostrata è incompleta, in quanto si consente solo la visualizzazione di numeri interi e stringhe. Nel listato successivo, relativo a «vsnprintf.c», si vedono diverse funzioni dichiarate in modo «statico», dato che servono esclusivamente a *vsnprintf()*.

Listato u166.20. «./05/lib/vsnprintf.c»

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

//
// Converte un intero senza segno di rango massimo in una stringa.
//
static size_t
uimaxtoa(uintmax_t integer, char *buffer, int base, int uppercase, size_t n)
{
    uintmax_t integer_copy = integer;
    size_t digits;
    int b;
    unsigned char remainder;

    for (digits = 0; integer_copy > 0; digits++)
    {
        integer_copy = integer_copy / base;
    }

    if (buffer == NULL && integer == 0) return 1;
    if (buffer == NULL && integer > 0) return digits;

    if (integer == 0)
    {
        buffer[0] = '0';
        buffer[1] = '\0';
        return 1;
    }

    if (n > 0 && digits > n) digits = n; // Sistema il numero massimo
                                        // di cifre.

    *(buffer + digits) = '\0'; // Fine della stringa.

    for (b = digits - 1; integer != 0 && b >= 0; b--)
    {
        remainder = integer % base;
        integer = integer / base;

        if (remainder <= 9)
        {
            *(buffer + b) = remainder + '0';
        }
        else
        {
            if (uppercase)
            {
                *(buffer + b) = remainder - 10 + 'A';
            }
            else
            {
                *(buffer + b) = remainder - 10 + 'a';
            }
        }
    }
}
```

1920

```
        *(buffer + b) = remainder - 10 + 'a';
    }
}

return digits;
}

//
// Converte un intero con segno di rango massimo in una stringa.
//
static size_t
imaxtoa(intmax_t integer, char *buffer, int base, int uppercase, size_t n)
{
    if (integer >= 0)
    {
        return uimaxtoa(integer, buffer, base, uppercase, n);
    }
    //
    // A questo punto c'è un valore negativo, inferiore a zero.
    //
    if (buffer == NULL)
    {
        return uimaxtoa(-integer, NULL, base, uppercase, n) + 1;
    }

    *buffer = '-'; // Serve il segno meno all'inizio.
    if (n == 1)
    {
        *(buffer + 1) = '\0';
        return 1;
    }
    else
    {
        return uimaxtoa(-integer, buffer+1, base, uppercase, n-1) + 1;
    }
}

//
// Converte un intero con segno di rango massimo in una stringa,
// mettendo il segno anche se è positivo.
//
static size_t
simaxtoa(intmax_t integer, char *buffer, int base, int uppercase, size_t n)
{
    if (buffer == NULL && integer >= 0)
    {
        return uimaxtoa(integer, NULL, base, uppercase, n) + 1;
    }

    if (buffer == NULL && integer < 0)
    {
        return uimaxtoa(-integer, NULL, base, uppercase, n) + 1;
    }
    //
    // A questo punto «buffer» è diverso da NULL.
    //
    if (integer >= 0)
    {
        *buffer = '+';
    }
    else
    {
        *buffer = '-';
    }

    if (n == 1)
    {
        *(buffer + 1) = '\0';
        return 1;
    }

    if (integer >= 0)
    {
        return uimaxtoa(integer, buffer+1, base, uppercase, n-1) + 1;
    }
    else
    {
        return uimaxtoa(-integer, buffer+1, base, uppercase, n-1) + 1;
    }
}

//
// Converte un intero senza segno di rango massimo in una stringa,
// provvedendo a sistemare anche l'allineamento.
//
static size_t
uimaxtoa_fill(uintmax_t integer, char *buffer, int base,
              int uppercase, int width, int filler, int max)
{
    if (max < 0) return 0; // «max» deve essere un valore positivo.

    size_t size_i = uimaxtoa(integer, NULL, base, uppercase, 0);
    size_t size_f;

    if (width > 0 && max > 0 && width > max) width = max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (size_i > abs(width))
    {
        return uimaxtoa(integer, buffer, base, uppercase, abs(width));
    }

    if (width == 0 && max > 0)
```

1921

```

    {
        return uimaxtoa (integer, buffer, base, uppercase, max);
    }

    if (width == 0)
    {
        return uimaxtoa (integer, buffer, base, uppercase, abs (width));
    }
    //
    // size_i <= abs (width).
    //
    size_f = abs (width) - size_i;

    if (width < 0)
    {
        // Allineamento a sinistra.
        uimaxtoa (integer, buffer, base, uppercase, 0);
        memset (buffer + size_i, filler, size_f);
    }
    else
    {
        // Allineamento a destra.
        memset (buffer, filler, size_f);
        uimaxtoa (integer, buffer + size_f, base, uppercase, 0);
    }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//
// Converte un intero con segno di rango massimo in una stringa,
// provvedendo a sistemare anche l'allineamento.
//
static size_t
imaxtoa_fill (intmax_t integer, char *buffer, int base,
              int uppercase, int width, int filler, int max)
{
    if (max < 0) return 0; // «max» deve essere un valore positivo.

    size_t size_i = imaxtoa (integer, NULL, base, uppercase, 0);
    size_t size_f;

    if (width > 0 && max > 0 && width > max) width = max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (size_i > abs (width))
    {
        return imaxtoa (integer, buffer, base, uppercase, abs (width));
    }

    if (width == 0 && max > 0)
    {
        return imaxtoa (integer, buffer, base, uppercase, max);
    }

    if (width == 0)
    {
        return imaxtoa (integer, buffer, base, uppercase, abs (width));
    }

    // size_i <= abs (width).

    size_f = abs (width) - size_i;

    if (width < 0)
    {
        // Allineamento a sinistra.
        imaxtoa (integer, buffer, base, uppercase, 0);
        memset (buffer + size_i, filler, size_f);
    }
    else
    {
        // Allineamento a destra.
        memset (buffer, filler, size_f);
        imaxtoa (integer, buffer + size_f, base, uppercase, 0);
    }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//
// Converte un intero con segno di rango massimo in una stringa,
// mettendo il segno anche se è positivo, provvedendo a sistemare
// l'allineamento.
//
static size_t
simaxtoa_fill (intmax_t integer, char *buffer, int base,
              int uppercase, int width, int filler, int max)
{
    if (max < 0) return 0; // «max» deve essere un valore positivo.

    size_t size_i = simaxtoa (integer, NULL, base, uppercase, 0);
    size_t size_f;

    if (width > 0 && max > 0 && width > max) width = max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (size_i > abs (width))
    {
        return simaxtoa (integer, buffer, base, uppercase, abs (width));
    }

    if (width == 0 && max > 0)

```

```

    {
        return simaxtoa (integer, buffer, base, uppercase, max);
    }

    if (width == 0)
    {
        return simaxtoa (integer, buffer, base, uppercase, abs (width));
    }
    //
    // size_i <= abs (width).
    //
    size_f = abs (width) - size_i;

    if (width < 0)
    {
        // Allineamento a sinistra.
        simaxtoa (integer, buffer, base, uppercase, 0);
        memset (buffer + size_i, filler, size_f);
    }
    else
    {
        // Allineamento a destra.
        memset (buffer, filler, size_f);
        simaxtoa (integer, buffer + size_f, base, uppercase, 0);
    }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//
// Trasferisce una stringa provvedendo all'allineamento.
//
static size_t
strtostr_fill (char *string, char *buffer, int width, int filler, int max)
{
    if (max < 0) return 0; // «max» deve essere un valore positivo.

    size_t size_s = strlen (string);
    size_t size_f;

    if (width > 0 && max > 0 && width > max) width = max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (width != 0 && size_s > abs (width))
    {
        memcpy (buffer, string, abs (width));
        buffer[width] = '\0';
        return width;
    }

    if (width == 0 && max > 0 && size_s > max)
    {
        memcpy (buffer, string, max);
        buffer[max] = '\0';
        return max;
    }

    if (width == 0 && max > 0 && size_s < max)
    {
        memcpy (buffer, string, size_s);
        buffer[size_s] = '\0';
        return size_s;
    }
    //
    // width != 0
    // size_s <= abs (width)
    //
    size_f = abs (width) - size_s;

    if (width < 0)
    {
        // Allineamento a destra.
        memset (buffer, filler, size_f);
        strncpy (buffer+size_f, string, size_s);
    }
    else
    {
        // Allineamento a sinistra.
        strncpy (buffer, string, size_s);
        memset (buffer+size_s, filler, size_f);
    }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//
// La funzione «vsprintf()»
//
int
vsprintf (char *restrict string, size_t n,
         const char *restrict format, va_list ap)
{
    if (n > INT_MAX) n = INT_MAX; // «n» non può essere superiore
    // a INT_MAX.

    //
    // Al massimo si producono "n-1" caratteri, + '\0'.
    // "n" viene usato anche come dimensione massima per le
    // stringhe interne, se non è troppo grande.
    //
    int f = 0;
    int s = 0;
    int remain = n - 1;

```

```

bool    specifier      = 0;
bool    specifier_flags = 0;
bool    specifier_width = 0;
bool    specifier_precision = 0;
bool    specifier_type  = 0;

bool    flag_plus      = 0;
bool    flag_minus     = 0;
bool    flag_space     = 0;
bool    flag_alternate = 0;
bool    flag_zero      = 0;

int     alignment;
int     filler;

intmax_t value_i;
uintmax_t value_ui;
char    +value_cp;

size_t  width;
size_t  precision;
size_t  str_size = n > 1024 ? 1024 : n;
char    width_string[str_size];
char    precision_string[str_size];
int     w;
int     p;

width_string[0] = '\0';
precision_string[0] = '\0';

while (format[f] != 0 && s < (n - 1))
{
    if (!specifier)
    {
        if (format[f] != '%')
        {
            string[s] = format[f];
            s++;
            remain--;
            f++;
            continue;
        }
        if (format[f] == '%' && format[f+1] == '%')
        {
            string[s] = '%';
            f++;
            f++;
            s++;
            remain--;
            continue;
        }
        if (format[f] == '%')
        {
            f++;
            specifier = 1;
            specifier_flags = 1;
            continue;
        }
    }

    if (specifier && specifier_flags)
    {
        if (format[f] == '+')
        {
            flag_plus = 1;
            f++;
            continue;
        }
        else if (format[f] == '-')
        {
            flag_minus = 1;
            f++;
            continue;
        }
        else if (format[f] == ' ')
        {
            flag_space = 1;
            f++;
            continue;
        }
        else if (format[f] == '#')
        {
            flag_alternate = 1;
            f++;
            continue;
        }
        else if (format[f] == '0')
        {
            flag_zero = 1;
            f++;
            continue;
        }
        else
        {
            specifier_flags = 0;
            specifier_width = 1;
        }
    }

    if (specifier && specifier_width)
    {
        for (w = 0; format[f] >= '0' && format[f] <= '9'

```

1924

```

        && w < str_size; w++)
        {
            width_string[w] = format[f];
            f++;
        }
        width_string[w] = '\0';

        specifier_width = 0;

        if (format[f] == '.')
        {
            specifier_precision = 1;
            f++;
        }
        else
        {
            specifier_precision = 0;
            specifier_type = 1;
        }
    }

    if (specifier && specifier_precision)
    {
        for (p = 0; format[f] >= '0' && format[f] <= '9'
            && p < str_size; p++)
        {
            precision_string[p] = format[f];
            p++;
        }
        precision_string[p] = '\0';

        specifier_precision = 0;
        specifier_type = 1;
    }

    if (specifier && specifier_type)
    {
        width = atoi (width_string);
        precision = atoi (precision_string);
        filler = ' ';
        if (flag_zero) filler = '0';
        if (flag_space) filler = ' ';
        alignment = width;

        if (flag_minus)
        {
            alignment = -alignment;
            filler = ' '; // Il carattere di riempimento
                        // non può essere zero.
        }

        if (format[f] == 'h' && format[f+1] == 'h')
        {
            if (format[f+2] == 'd' || format[f+2] == 'i')
            {
                // signed char, base 10.
                value_i = va_arg (ap, int);
                if (flag_plus)
                {
                    s += simaxtoa_fill (value_i, &string[s], 10, 0,
                                        alignment, filler, remain);
                }
                else
                {
                    s += imaxtoa_fill (value_i, &string[s], 10, 0,
                                        alignment, filler, remain);
                }
                f += 3;
            }
            else if (format[f+2] == 'u')
            {
                // unsigned char, base 10.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                                    alignment, filler, remain);
                f += 3;
            }
            else if (format[f+2] == 'o')
            {
                // unsigned char, base 8.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                    alignment, filler, remain);
                f += 3;
            }
            else if (format[f+2] == 'x')
            {
                // unsigned char, base 16.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                    alignment, filler, remain);
                f += 3;
            }
            else if (format[f+2] == 'X')
            {
                // unsigned char, base 16.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                                    alignment, filler, remain);
                f += 3;
            }
            else if (format[f+2] == 'b')
            {
                // unsigned char, base 2 (estensione).

```

1925



```

        value_ui = va_arg (ap, unsigned int);
        s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                           alignment, filler, remain);
        f += 3;
    }
    else // Specificatore errato:
    {
        f += 2;
    }
}
else if (format[f] == 'h')
{
    if (format[f+1] == 'd' || format[f+1] == 'i')
    {
        // short int, base 10.
        value_i = va_arg (ap, int);
        if (flag_plus)
        {
            s += simaxtoa_fill (value_i, &string[s], 10, 0,
                               alignment, filler, remain);
        }
        else
        {
            s += imaxtoa_fill (value_i, &string[s], 10, 0,
                              alignment, filler, remain);
        }
        f += 2;
    }
    else if (format[f+1] == 'u')
    {
        // unsigned short int, base 10.
        value_ui = va_arg (ap, unsigned int);
        s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                           alignment, filler, remain);
        f += 2;
    }
    else if (format[f+1] == 'o')
    {
        // unsigned short int, base 8.
        value_ui = va_arg (ap, unsigned int);
        s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                           alignment, filler, remain);
        f += 2;
    }
    else if (format[f+1] == 'x')
    {
        // unsigned short int, base 16.
        value_ui = va_arg (ap, unsigned int);
        s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                           alignment, filler, remain);
        f += 2;
    }
    else if (format[f+1] == 'X')
    {
        // unsigned short int, base 16.
        value_ui = va_arg (ap, unsigned int);
        s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                           alignment, filler, remain);
        f += 2;
    }
    else if (format[f+1] == 'b')
    {
        // unsigned short int, base 2 (estensione).
        value_ui = va_arg (ap, unsigned int);
        s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                           alignment, filler, remain);
        f += 2;
    }
    else // Specificatore errato:
    {
        f += 1;
    }
}
//
// Il tipo «long long int» non c'è, perché il compilatore
// GNU C, per poter eseguire le divisioni e il calcolo del
// resto, ha bisogno delle funzioni di libreria
// «_udivdi3()» e «_umoddi3()».
//
else if (format[f] == 'l')
{
    if (format[f+1] == 'd' || format[f+1] == 'i')
    {
        // long int base 10.
        value_i = va_arg (ap, long int);
        if (flag_plus)
        {
            s += simaxtoa_fill (value_i, &string[s], 10, 0,
                               alignment, filler, remain);
        }
        else
        {
            s += imaxtoa_fill (value_i, &string[s], 10, 0,
                              alignment, filler, remain);
        }
        f += 2;
    }
    else if (format[f+1] == 'u')
    {
        // Unsigned long int base 10.
        value_ui = va_arg (ap, unsigned long int);
        s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                           alignment, filler, remain);

```

```

        f += 2;
    }
    else if (format[f+1] == 'o')
    {
        // Unsigned long int base 8.
        value_ui = va_arg (ap, unsigned long int);
        s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                           alignment, filler, remain);
        f += 2;
    }
    else if (format[f+1] == 'x')
    {
        // Unsigned long int base 16.
        value_ui = va_arg (ap, unsigned long int);
        s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                           alignment, filler, remain);
        f += 2;
    }
    else if (format[f+1] == 'X')
    {
        // Unsigned long int base 16.
        value_ui = va_arg (ap, unsigned long int);
        s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                           alignment, filler, remain);
        f += 2;
    }
    else if (format[f+1] == 'b')
    {
        // Unsigned long int base 2 (estensione).
        value_ui = va_arg (ap, unsigned long int);
        s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                           alignment, filler, remain);
        f += 2;
    }
    else // Specificatore errato:
    {
        f += 1;
    }
}
else if (format[f] == 'j')
{
    if (format[f+1] == 'd' || format[f+1] == 'i')
    {
        // intmax_t base 10.
        value_i = va_arg (ap, intmax_t);
        if (flag_plus)
        {
            s += simaxtoa_fill (value_i, &string[s], 10, 0,
                               alignment, filler, remain);
        }
        else
        {
            s += imaxtoa_fill (value_i, &string[s], 10, 0,
                              alignment, filler, remain);
        }
        f += 2;
    }
    else if (format[f+1] == 'u')
    {
        // uintmax_t base 10.
        value_ui = va_arg (ap, uintmax_t);
        s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                           alignment, filler, remain);
        f += 2;
    }
    else if (format[f+1] == 'o')
    {
        // uintmax_t base 8.
        value_ui = va_arg (ap, uintmax_t);
        s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                           alignment, filler, remain);
        f += 2;
    }
    else if (format[f+1] == 'x')
    {
        // uintmax_t base 16.
        value_ui = va_arg (ap, uintmax_t);
        s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                           alignment, filler, remain);
        f += 2;
    }
    else if (format[f+1] == 'X')
    {
        // uintmax_t base 16.
        value_ui = va_arg (ap, uintmax_t);
        s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                           alignment, filler, remain);
        f += 2;
    }
    else if (format[f+1] == 'b')
    {
        // uintmax_t base 2 (estensione).
        value_ui = va_arg (ap, uintmax_t);
        s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                           alignment, filler, remain);
        f += 2;
    }
    else // Specificatore errato:
    {
        f += 1;
    }
}
else if (format[f] == 'z')

```

```

{
  if (format[f+1] == 'd'
      || format[f+1] == 'i'
      || format[f+1] == 'i')
  {
    // size_t base 10.
    value_ui = va_arg (ap, unsigned long int);
    s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                       alignment, filler, remain);
    f += 2;
  }
  else if (format[f+1] == 'o')
  {
    // size_t base 8.
    value_ui = va_arg (ap, unsigned long int);
    s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                       alignment, filler, remain);
    f += 2;
  }
  else if (format[f+1] == 'x')
  {
    // size_t base 16.
    value_ui = va_arg (ap, unsigned long int);
    s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                       alignment, filler, remain);
    f += 2;
  }
  else if (format[f+1] == 'X')
  {
    // size_t base 16.
    value_ui = va_arg (ap, unsigned long int);
    s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                       alignment, filler, remain);
    f += 2;
  }
  else if (format[f+1] == 'b')
  {
    // size_t base 2 (estensione).
    value_ui = va_arg (ap, unsigned long int);
    s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                       alignment, filler, remain);
    f += 2;
  }
  else // Specificatore errato:
  {
    f += 1;
  }
}
else if (format[f] == 't')
{
  if (format[f+1] == 'd' || format[f+1] == 'i')
  {
    // ptrdiff_t base 10.
    value_i = va_arg (ap, long int);
    if (flag_plus)
    {
      s += simaxtoa_fill (value_i, &string[s], 10, 0,
                         alignment, filler, remain);
    }
    else
    {
      s += imaxtoa_fill (value_i, &string[s], 10, 0,
                        alignment, filler, remain);
    }
    f += 2;
  }
  else if (format[f+1] == 'u')
  {
    // ptrdiff_t base 10, senza segno.
    value_ui = va_arg (ap, unsigned long int);
    s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                       alignment, filler, remain);
    f += 2;
  }
  else if (format[f+1] == 'o')
  {
    // ptrdiff_t base 8, senza segno.
    value_ui = va_arg (ap, unsigned long int);
    s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                       alignment, filler, remain);
    f += 2;
  }
  else if (format[f+1] == 'x')
  {
    // ptrdiff_t base 16, senza segno.
    value_ui = va_arg (ap, unsigned long int);
    s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                       alignment, filler, remain);
    f += 2;
  }
  else if (format[f+1] == 'X')
  {
    // ptrdiff_t base 16, senza segno.
    value_ui = va_arg (ap, unsigned long int);
    s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                       alignment, filler, remain);
    f += 2;
  }
  else if (format[f+1] == 'b')
  {
    // ptrdiff_t base 2, senza segno.
    value_ui = va_arg (ap, unsigned long int);
    s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                       alignment, filler, remain);

```

```

alignment, filler, remain);
    f += 2;
  }
  else // Specificatore errato:
  {
    f += 1;
  }
}
if (format[f] == 'd' || format[f] == 'i')
{
  // int base 10.
  value_i = va_arg (ap, int);
  if (flag_plus)
  {
    s += simaxtoa_fill (value_i, &string[s], 10, 0,
                       alignment, filler, remain);
  }
  else
  {
    s += imaxtoa_fill (value_i, &string[s], 10, 0,
                      alignment, filler, remain);
  }
  f += 1;
}
else if (format[f] == 'u')
{
  // unsigned int base 10.
  value_ui = va_arg (ap, unsigned int);
  s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                     alignment, filler, remain);
  f += 1;
}
else if (format[f] == 'o')
{
  // Unsigned int base 8.
  value_ui = va_arg (ap, unsigned int);
  s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                     alignment, filler, remain);
  f += 1;
}
else if (format[f] == 'x')
{
  // unsigned int base 16.
  value_ui = va_arg (ap, unsigned int);
  s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                     alignment, filler, remain);
  f += 1;
}
else if (format[f] == 'X')
{
  // unsigned int base 16.
  value_ui = va_arg (ap, unsigned int);
  s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                     alignment, filler, remain);
  f += 1;
}
else if (format[f] == 'b')
{
  // unsigned int base 2 (estensione).
  value_ui = va_arg (ap, unsigned int);
  s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                     alignment, filler, remain);
  f += 1;
}
//else if (format[f] == 'c')
// {
//   // unsigned char.
//   value_ui = va_arg (ap, unsigned int);
//   s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
//                       alignment, filler, remain);
//   f += 1;
// }
else if (format[f] == 'c')
{
  // unsigned char.
  value_ui = va_arg (ap, unsigned int);
  string[s] = (char) value_ui;
  s += 1;
  f += 1;
}
else if (format[f] == 's')
{
  // string.
  value_cp = va_arg (ap, char *);
  filler = ' ';
  s += strtosttr_fill (value_cp, &string[s], alignment,
                      filler, remain);
  f += 1;
}
}
else // Specificatore errato:
{
  ;
}
//
// Fine dello specificatore.
//
width_string[0] = '\0';
precision_string[0] = '\0';

specifier      = 0;
specifier_flags = 0;
specifier_width = 0;

```

```
    specifier_precision = 0;
    specifier_type      = 0;

    flag_plus          = 0;
    flag_minus         = 0;
    flag_space         = 0;
    flag_alternate     = 0;
    flag_zero          = 0;
}
}
string[s] = '\0';
return s;
}
```

#### Listato ul66.21. './05/lib/snprintf.c'

```
#include <stdio.h>
int
snprintf (char *restrict string, size_t n, const char *restrict format, ...)
{
    va_list ap;
    va_start (ap, format);
    return vsnprintf (string, n, format, ap);
}
```