Input/Output

- **int fread(void *ptr, int size, int numitems, FILE *stream)**
  - Fread "reads" numitems objects of specified size into the block ptr, from the stream. (Actually, "read" may just mean "copy from the stream buffer".). It returns the number of items "read".
- **int fwrite(void *ptr, int size, int numitems, FILE *stream)**
  - Fwrite "writes" numitems objects of size specified from the address ptr to the stream. (Actually, "write" may just mean "copy to the stream buffer".). It returns the number of items "written".
- **int getc(FILE *stream)**
- **int getchar()**
  - Getc is a macro that "reads" one character from the stream. Getchar is just getc(stdin). They return EOF on end of file. (EOF is usually defined as -1. Thus the procedures return a non-character value on end of file.).
- **char *gets(char *s)**
  - Gets "reads" the rest of the line from stdin, and stores it, null terminated, in s, which it returns as its result. The newline is consumed.

X-printf

int printf(char *format, ...)
int fprintf(FILE *stream, char *format, ...)
int sprintf(char *s, char *format, ...)

- These functions convert their arguments into a string according to the format, and "write" the resultant string out to the standard output, the specified stream, or the specified string (null terminated), respectively.
- The format string contains text (which is just transferred over), and a conversion specification for each argument. A conversion specification is of the form

% [–] [fieldwidth] [.[precision]] [l] [duoxfegcs]

The meaning is as follows:

- – Left justify the result in the field.
- fieldwidth
  - decimal digit string indicating the minimum size of the text created.
  - The text is padded with blanks to bring it up to the specified field width, or with a '0', if the field width begins with a 0.
- precision
  - decimal digit string indicating the number of digits to appear after the decimal point in e and f format, or the maximum number of characters to be printed from a string.
- L The argument is a long integer.

Printf(2)

duoxfegcs

The argument is formatted according to the character specified.

- d (decimal), u (unsigned decimal), o (octal), x (hexadecimal) for integer value,
- f (fixed point), e (exponent), g (fixed or exponent, as appropriate) for floating point value,
- c (ASCII character),
- s (ASCII string).

A format character is essential. The other parts of the specification are optional.

To print %, the character is doubled, as in %%.

The return value is the number of characters printed.

An example of a printf statement is

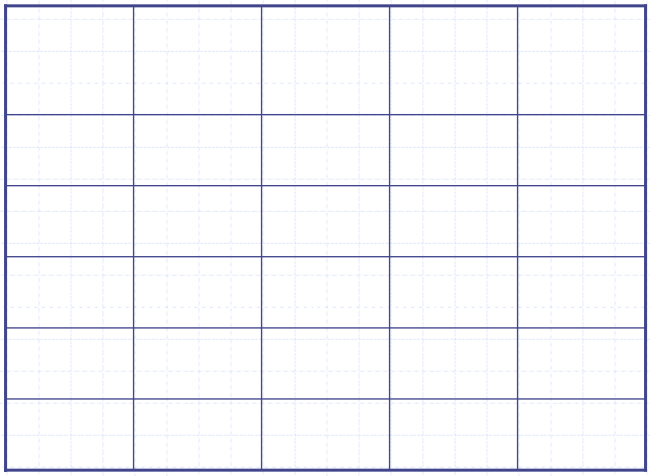
printf("s = %s, n = %d in decimal, and %o in octal\n", s, n, n);

Example sheet 1

Memory/ byte	String 1 /hex
0x1001	S / 0x53
0x1002	T / 0x54
0x1003	O / 0x50
0x1004	P / 0x51
0x1005	NULL/0x00

Memory/ byte	String 2 /hex
0x1006	H /0x48
0x1007	E /0x45
0x1008	L /0x4C
0x1009	M /0x4D
0x100a	NULL/0x00

Example sheet 2



Alphanumerical table

Alpha Computer Architecture August 26, 2002

Textual Characters:

The characters with ASCII value 0x21 to 0x7e represent textual characters.

The full ASCII character set is as follows

00	NUL	01	SOH	02	STX	03	ETX								
04	EOT	05	ENQ	06	ACK	07	BEL (\a)								
08	BS (\b)	09	HT (\t)	0A	LF (\n)	0B	VT (\v)								
0C	FF (\f)	0D	CR (\r)	0E	SO	0F	SI								
10	DLE	11	DC1	12	DC2	13	DC3								
14	DC4	15	NAK	16	SYN	17	ETB								
18	CAN	19	EM	1A	SUB	1B	ESC								
1C	FS	1D	GS	1E	RS	1F	US								
20	SP	21	!	22	"	23	#								
24	$	25	%	26	&	27	'								
28	(	29	)	2A	*	2B	+								
2C	,	2D	-	2E	.	2F	/								
30	0	31	1	32	2	33	3								
34	4	35	5	36	6	37	7								
38	8	39	9	3A	:	3B	;								
3C	<	3D	=	3E	>	3F	?								
40	@	41	A	42	B	43	C								
44	D	45	E	46	F	47	G								
48	H	49	I	4A	J	4B	K								
4C	L	4D	M	4E	N	4F	O								
50	P	51	Q	52	R	53	S								
54	T	55	U	56	V	57	W								
58	X	59	Y	5A	Z	5B	[								
5C	\	5D	]	5E	^	5F	_								
60	`	61	a	62	b	63	c								
64	d	65	e	66	f	67	g								
68	h	69	i	6A	j	6B	k								
6C	l	6D	m	6E	n	6F	o								
70	p	71	q	72	r	73	s								
74	t	75	u	76	v	77	w								
78	x	79	y	7A	z	7B	{								
7C	|	7D	}	7E	~	7F	DEL								

Postfix-Prefix

a++, a--Postfix increment and decrement operators. The operand is incremented or decremented (by 1 for integer variables, by the size of the object pointed to for pointer variables). The value returned is the initial value of the variable.

++a, --aPrefix increment and decrement operators. The operand is incremented or decremented (by 1 for integer variables, by the size of the object pointed to for pointer variables). The value returned is the final value of the variable.

Structures

Structure types

The C structure type is equivalent to the Pascal record type, and similar to a Java class with only instance fields. A structure is a compound object composed of named component fields, packed together side by side. For example, we could declare the structure type Node, representing a node of a linked list

```
struct Node {
    int value;
    struct Node *next;
    };
```

then declare a variable of that type

```
struct Node node;          //      Actual structure
struct Node *listPtr; //      Pointer to a structure
```

Note that the declaration of a structure variable allocates space for the structure.

The variable is the structure, not a pointer to a structure. As a consequence

```
struct Node {
    int value;
    struct Node next; // Illegal!
    };
```

is illegal, because it a value of this type would require infinite space.

Structures (2)

To access the components of a structure, we use the '.' notation. If a is a structure variable, and b is a field of the structure, then "a.b" represents the appropriate field b within a.

```
struct Complex { int x, y; };
struct Complex one;
one.x = 1;
one.y = 0;
```

Pointers to structures are commonly used in C.

```
struct Node *listPtr;
```

The accessing of fields by "(*listPtr).value" occurs so often, that a special equivalent notation has been provided,

namely "listPtr->value".

Function types

Not only can we declare functions in C, we can also declare function variables and parameters (really pointers to functions), and dynamically assign (a pointer to) a function to the variable or parameter.

For any non array type, we can have a function returning that type.

To formally declare a function, when not specifying its body, write something like int f(int x, int y, char *s);

In a formal declaration of a function, it is possible to omit the names of the formal parameters int f(int, int, char *);

To invoke a function, write the function name, followed by the actual parameters, enclosed in parentheses, and separated by commas. For example f(a, b, &c)

The parentheses are essential, even if there are no parameters.

Declaration

Typedef declarations

It is also possible to declare type names, by a typedef declaration of the form

typedef Type DeclaratorSequence;

The identifiers appearing in the declarator sequence are declared to be equal to the type they would correspond to if the above was an ordinary declaration, without the typedef keyword:

typedef struct Node *List; declares List to be the type corresponding to the type pointer to struct Node. The typedef name can then be used anywhere a type can occur. For example List p;

Function declarations

A full declaration of a function is of the form

Type Declarator (ParameterDeclSequence) {
Body
}

A formal declaration of a function is of the form

Type Declarator (ParameterDeclSequence);

Functions have to be declared before they are used (for example, in a #include file).

Programme examples

The following function invokes getchar() to read in a line of text, and store in a buffer. If the line is too long, it still reads the line, but discards the characters that cannot fit in the buffer.

```
char *readLine( char *s, int max ) {
    register int i = 0;
    register int c;
    while ( TRUE ) {
            c = getchar();
            if ( c < 0 || c == '\n' )
                    break;
            if ( i < max )
                    s[ i ] = c;
            i++;
            }
    if ( i > max )
            i = max;
    s[ i ] = '\0';
    if ( c < 0 )
            return NULL;
    else
            return s + i;
    }
```

Storage class

The register storage class specifier

Indicates to the compiler that the object should be stored in a machine register.

Typically specified for heavily used variables to improve performance by minimizing access time.

Because of the limited size and number of registers available the request may not be granted.

In this case the object is treated as belongingto the storage class specifier auto.

An object having the register storage class specifier must be defined within a block or declared as a parameter to a function.

Restrictions to the register storage class specifier:

1. You cannot use pointers to reference register storage class specified objects.
2. You cannot use the register storage class specifier when declaring objects in global scope.
3. A register does not have an address. You cannot apply the address operator (&) to a register variable.

Matching pattern (1)

```
/*
match1.c

finds lines containing a matching pattern

*/
//------------------------------

#include <stdio.h>
#define MAXLINE 1000                //maximum line length
//------------------------------

int getline(char line[], int max);
int strindex(char source[],char searchfor[]);

char pattern[] = "ere";             // pattern to search for

//------------------------------
```

Matching pattern (2)

```
/* find all lines matching pattern */
int main(){

    char line[MAXLINE];
    int found;

    if (getline(line,MAXLINE) > 0 ){
        if ((found = strindex(line,pattern))>=0) {
            printf("%s occurs at position %d in ",
    pattern, found+1);
            printf("\"%s\"", line);
        }
    }
    return 0;
}
```


Matching pattern (3)-getline

```
//------------------------------

/* getline: get line into s, return length*/
int getline(char s[], int lim){

    int c, i;

    i = 0;

    while(--lim > 0 && (c=getchar())!='\n')
        s[i++] = c;
    s[i] = '\0';
    return i;
}
```

Matching pattern (4)-strindex

```
//-------------------------------
/* strindex: return index of t in s, -1 if none */
int strindex(char s[], char t[]){

    int i, j, k;

    for (i=0; s[i]!='\0'; i++){

            for (j=i, k=0; t[k]!='\0' && s[j]==t[k]; j++, k++)
                    ;
            if (k>0 && t[k]=='\0')
                    return i;

    }//for (i=0; s[i]!='\0'; i++){

    return -1;
}
```