Block

{
definitions-and-declarations (optional)
statement-list
}

Used for grouping statements into an executable unit, as function body, and to restrict identifier visibility.

No semicolon after closing brace!

Statements

Null statement: denoted by semicolon

Expression statement: expression followed by semicolon.

if statement

for Loops

while Loops

do Loops

Be careful with if

Using Null statement and Compound statement to avoid dangling **else** problem

What is the means of

if (e_1) if (e_2) s_a else s_b

More Control Flow

continue Statement

Used to skip remaining portion of current iteration

break Statement

Used to immediately exit from enclosing loop

goto Statement can be used as multilevel break sometime

switch Statement

Logical Operators

Logical Operators: ||, &&, !
&& and || have lower precedence than ! and && above ||

Non-zero numerical value is true and 0 is false.

Example: if (!valid) same as if (valid == 0)

Think about what does !!x do?

Arithmetic Operators

Operators: + - * / % (modulus)

Unary ++ and −− have hightest precedence.

*, / and % are next and binary + and − are lowest.

Associativity left to right.

Relational Operators

Relational operators have lower precedence than arithmetic operators.

>, >=, < and <= have the same precedence.

== and != are lower

Examples: $i < lim - 1$ $\qquad a > b == c > d$

Assignment Operators

Assignment operators have lowest precedence except comma operator, right-to-left associativity.

Examples: a=b=c+d

When variable on left side of assignment is repeated immediately on right, as in i = i * 9, can be write as i *= 9

$expr_1\ op = expr_2$ is equivalent to $expr_1 = expr_1\ op\ (expr_2)$

Example: $x *= \ y - 1$ means $x = x * (y - 1)$ not $x = x * y - 1$

Pointer in C

- What is pointer?
- Pointer and Arrays
- Pointer Arithmetic
- Pointer and Strings
- Multidimensional Arrays
- Pointer and functions
- Common Pointer Pitfalls

What is pointer? (I)

- Variable
  A simple variable in a program is stored in a certain number of bytes at a particular memory location, or address, in the machine. For example, on SUN the size of an integer variable is 4 bytes, and that of a long integer is also 4 bytes.
- Pointer
  Pointers are used in programs to access memory and manipulate address.

What is pointer? (II)

- address of variable
  The address operator & can be used to get the address of a variable in memory where it store its value. Addresses are a set of values that similar to integer and can be manipulated.
- pointer variable
  - it is a variable itself.
  - it accept address of memory as its value
  - it is declared by giving it a type and a name with the asterisk

What is pointer? (III)

The unary operator & gives the **address of a variable**.

The indirection or dereference operator * gives the **contents of an object pointed to by a pointer**.

What is pointer? (Example)

- address of variable int a, b;
  &a, &b
- pointer variable int *p, *t;
  p=&a;
  t=p;

What is pointer? (NULL pointer)

A null pointer is a special pointer value that is known not to point anywhere.

What this means that no other valid pointer, to any other variable or array cell or anything else, will ever compare equal to a null pointer.

In fact, NULL is a preprocessor macro which typically has the value, or replacement text, 0.

One-dimensional Array

A type followed by an identifier with a bracketed constant integral expression (k) is represent a contiguous memory of k elements of that type.

int a[size];

A single element of array a can be accessed by $a[i]$ where $0 <= i <= n-1$.

Relation between arrays and pointers

- an array name itself is an address or pointer value.
- pointer and array can be subscripted.
- pointer and array use similar way to access memory
- array name can only be view as a CONSTANT pointer

Relation between arrays and pointers (Example)

int my_array[ ] = { 1, 23, 17, 4, -5, 100 };
int *p;
p = my_array;

- size of the array
- memory allocation
- address of the first item
- illegal expression my_array = p;

Pointer Arithmetic

If p is a pointer variable of type a, then p + 1 yield the correct address for the next position of type a.

int *pi,a[10]; chat *pc, c[10];
pi = a;
pc = c;
pi ++;
pc ++;

Pointers and Strings

In C, strings are one-dimensional arrays of characters terminated with a binary zero character (written as '\0').

char my_string[40]="Chalmers"; char *p="Chalmers";

Pointers and Strings (Example)

```
#include <stdio.h>

char strA[80] = "A string to be used for demonstration";
char strB[80];

int main(void)
{

    char *pA;     /* a pointer to type character */
    char *pB;     /* another pointer to type character */
    puts(strA);   /* show string A */
    pA = strA;    /* point pA at string A */
    puts(pA);     /* show what pA is pointing to */
```

```
    pB = strB;    /* point pB at string B */
    putchar('\n');
    while(*pA != '\0')
    {
        *pB++ = *pA++;
    }
    *pB = '\0';
    puts(strB);          /* show strB on screen */
    return 0;
}
```


Pointers and Strings (strcpy version I)

```
char *my_strcpy(char dest[], char source[])
{
    int i = 0;
    while (source[i] != '\0')
    {
        dest[i] = source[i];
        i++;
    }
    dest[i] = '\0';
    return dest;
}
```


Pointers and Strings (strcpy version II)

```
char *my_strcpy(char *destination, char *source)
{
    char *p = destination;
    while (*source != '\0')
    {
        *p++ = *source++;
    }
    *p = '\0';
    return destination;
}
```


Multidimensional Arrays

Multidimensional arrays can be view as arrays of arrays.

Add array dimension with add a bracket pair in definition. Examples:

int a[100]; int b[2][7]; int c[5][6][8];

Multidimensional arrays (Example)

int a[3][5];

```
Expressions equivalent to a[i][j]
---------------------------------
*(a[i] + j)
(*(a+i))[j]
*((*(a+i)) + j)
*(&a[0][0] + 5 * i + j)
```


Arrays of Pointers and Pointer to Array

In C, array elements can be of any type, including pointer.
In C, pointer can be ”pointed at” any type of data object, including arrays.

These two are both two-dimensional arrays.

int (*p1d)[10]; // Pointer to Array

int *p1d[10]; // Arrays of Pointers

Arrays of Pointers and Pointer to Array (Example)

```
int (*rptr)[COLS];
int main(void)
{
    int nrows = 10;
    int row, col;
    rptr = malloc(nrows * COLS * sizeof(int));
    for (row = 0; row < nrows; row++)
    {
        for (col = 0; col < COLS; col++)
            rptr[row][col] = 17;
    }
    return 0;
}
```


Pointer and functions

- pass arguments to function by value or by address
- return pointer from function
- pointer of function

Pointer and functions (Example I)

```
swap(a, b);

swap(&a, &b);

void swap(int *px, int *py)
{
   int temp;
   temp = *px; /* contents of pointer */
   *px = *py;
   *py = temp;
}
```


Pointer and functions (Example I)

int f(int);
int (*pf) (int) = &f;

Now f() returns an int and takes one int as a parameter.

ans = f(5);
ans = pf(5);

which are equivalent. Read the man page of qsort.

Common Pointer Pitfalls

- Not assigning a pointer to memory address before using it
- Illegal indirection
- Not checking a pointer return by a function before using it

Common Pointer Pitfalls (Example I)

```
int *x;
*x = 100;
*x = (int *) malloc(100);
*x = 100;
```

Common Pointer Pitfalls (Example II)

```
int *x;
x = (int *) malloc(100);
        if ( x == NULL)
        {
          printf("Error: Out of Memory \n");
          exit(1);
        }

*x = 'y';
```


Laboratory

Write a ftp server and a ftp client support the following command:

- ls (gives a listing of the current working directory on the server machine)
- cd <path> (changes the current working directory on the server machine to ¡path¿)
- get <filename> (sends the file ¡filename¿ from the server machine to the client machine).
- put <filename> (sends the file ¡filename¿ from the client machine to the server machine).
- quit (disconnects from the server and terminates the client)

Laboratory (Continue)

Your server program must be able to serve multiple clients simultaneously!!

Your system must be robust. (Cope with input of syntactically erroneous commands by the user, attempts to do cd to directories that do not exist, attempt to put/get non-existing files, etc.)

You need to use socket, strings and pipe library. Try to use Makefile to organise and compile your source code.