Lecture[1]

# C FUNDAMENTALS

# Agenda

- Data type and storage
- Expression
- Control flow

# Agenda

- Data type and storage
- Expression
- Control flow

# Data Type

- C has the following simple data types:

| Data type | C code | Size in bytes | Range |
|---|---|---|---|
| **Char or signed char** | char | 1 | -128 to 127 |
| **Unsigned char** | unsigned char | 1 | 0 to 255 |
| **int or signed int** | int | at least 2 | -32768 to 32767 |
| **Unsigned int** | unsigned int | at least 2 | 0 to 65535 |
| **Short int or signed short int** | short | 2 | -32768 to 32767 |
| **Unsigned short int** | unsigned short | 2 | 0 to 65535 |
| **Long int or Signed long int** | long | at least 4 | -2147483648 to 2147483647 |
| **Unsigned long int** | unsigned long | at least 4 | 0 to 4294967295 |
| **float** | float | 4 | 3.4E-38 to 3.4E+38 |
| **double** | double | 8 | 1.7E-308 to 1.7E+308 |
| **Long double** | long double | 10* | 3.4E-4932 to 1.1E+4932 |

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

int main()
{

    int b=3;
    //print out the value of b
    printf("the size of b is %u",sizeof(b));
    return 0;
}
```

```
the size of b is 4
```

On 32-bit machine is usually 4 bytes (32bits)

# Unsigned int

- How to store an unsigned int
  - All bits are used to store value

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

$$1111_2 = 15_{10}$$

  - 2 Bytes = 16 bits
  - $0 \sim 2^{16} - 1 = 65535$

# int with sign

- Need to both represent positive and negative int
- Sign and magnitude (原码)
  - –Use highest bit for sign: 0 -> +, 1 -> -
  - –Other bits are used to store value

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

$1111_2 = 15_{10}$

$1000111_2 = -15_{10}$

  - –But
    - What about 0? Two representations
    - 15+(-15)=?

# One's-Complement for signed numbers

- One's-Complement (二进制反码)
  - –Use highest bit for sign: 0 -> +, 1 -> -
  - –Positive number X is encode of X
  - –Negative number X is encode ~X and highest bit is 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

$15_{10}$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

$-15_{10}$

# Two's-Complement for signed numbers

- Two's-Complement (二进制补码)
  - Use highest bit for sign: 0 -> +, 1 -> -
  - Positive number X is encode of X
  - Negative number X is encode of $2^n$+X and highest bit is 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

$15_{10}$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

—

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

(方法2：反码+1)

$-15_{10}$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

# Two's-Complement for signed numbers

- Only one 0
- $-2^{n-1} \sim 2^{n-1}-1$
- The order is respected
- Arithmetic calculation

| Two's complement | Decimal |
| --- | --- |
| [0]111 | +7 |
| [0]110 | +6 |
| …… | …… |
| [0]001 | +1 |
| [0]000 | 0 |
| [1]111 | -1 |
| …… | …… |
| [1]001 | -7 |
| [1]000 | -8 |

# Floating types

float single-precision floating-point
double double-precision floating-point
long double extended-precision floating-point

| Type | Smallest Positive Value | Largest Value | Precision | Size |
|---|---|---|---|---|
| float | $1.17*10^{-38}$ | $3.40*10^{38}$ | 7 digits | 4 Bytes |
| double | $2.22*10^{-308}$ | $1.79*10^{308}$ | 15 digits | 8 Bytes |

```
float x;
scanf("%f", &x);
printf("%f", x);
```

```
double x;
scanf("%lf", &x);
printf("%f", x);
```

```
long double x;
scanf("%Lf", &x);
printf("%Lf", x);
```

# How to represent a float

- How to represent a decimal fraction in binary?
  - $10.125 \rightarrow 1010.001_2$
- Formatted form
  - $1010.001_2 = 0.1010001_2 \times 2^{100_2}$
- Encoding of float
  - **$N = (-1)^S \times 0.M \times 2^E$**

| S | E | M |
|---|---|---|
| 0 | 000100 | 001010001 |

10.125 in 16 bits

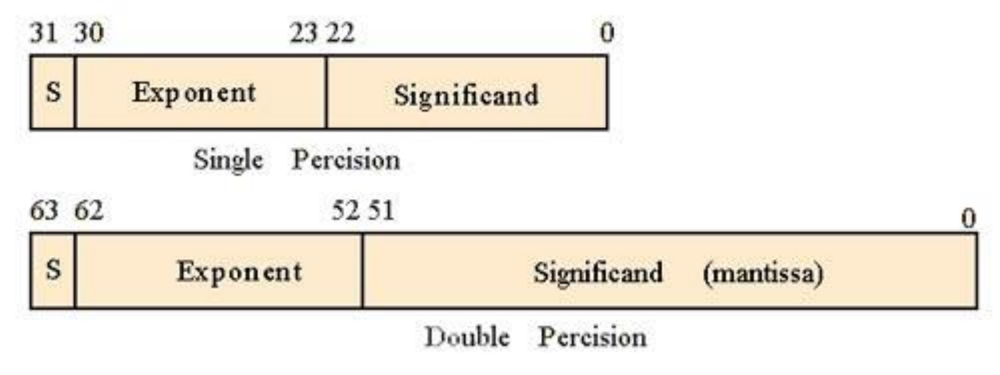
# IEEE 754 float

| S | E | M |
|---|---|---|
| 1 bit | 8 bits | 23bits |

- M is consider as 0.1xxxx->1.xxxx
  - Save one bit
- E use E= $E_{real}$+127
- **$N = (-1)^S \times 1.M \times 2^{E-127}$**
- The range is
  - Smallest value: E=1， M=0, $N_{min}=1.0 \times 2^{1-127}=2^{-126}$
  - Biggest value： E=254， M=11…1,
    $Nmax=1.11\ldots1 \times 2^{254-127}=(2-2^{-23}) \times 2^{127}$

# Float/double precision

- Float and double are not precise
  - –The storage is not accurate.
  - –Floating operation uses specific hardware
- Use double if no specific requirement



# floating types

- What’s the height of LBJ
- How 6 feet 9 inches 2.06m?
- Meter = (foot +inch/12)*0.3048
- Write a program

```
int main()
{
    printf(“input foot and inch:\n”);
    int foot;
    int inch;
    …
    return 0;
}
```

# Character Types

- Char is also an integer type
  - 'a', '1'
  - %c in scanf and printf
  - ASCII code
    - '1'的ASCII编码是49，
    - 当变量a==49，a的值就是'1'

```
char ch;
int i;
i = 'a';            // i is now ‘a’, its value is 97
ch = 65;            // ch is now ‘A’ , its value is 65
ch = ch + 1;        // ch is now ‘B’, its value is 66
ch++;               //ch is now ‘C’, its value is 67


for(ch = 'A'; ch <= 'Z'; ch++)
{
        printf("%c\n",ch);
}
```

# Boolean

- Boolean: {true, false}
- There is no Boolean data type in C
- Any integer ≠ 0 is considered true
- 0 is considered false

```
int a= 3;
int b = !a; // b=0
```

# Constants int

- Integer constants:
  - Decimal: 11,12345
- Declare a named constant:
  - Using **const** keyword

```
const int PAIR=2
```

  - Using **#define** pre-processor directive

```
#define PAIR 2
```

```
int main()
{
    int n;
    const int TIMES = 2;
    scanf(“%d”,&n);
    printf(“Output is %d\n”,TIMES*n);
}
```

# Constants

- Character constants
- String constants
  - "I am a string“
  - Character array
  - always null (‘\0’) terminated.
  - (see <string.h> for string functions)

# Agenda

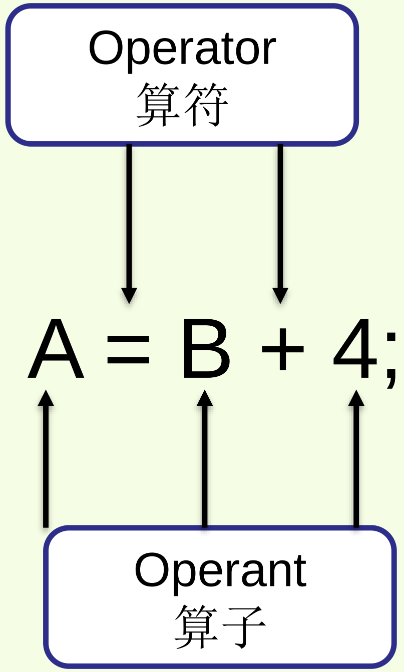
- Data type and storage
- Expression
- Control flow

# Agenda

- Data type and storage
- Expression
- Control flow

# Expressions

- Arithmetic operator:
  - +,-,*,/,%
- Relation operators:
  - <,>,<=,>=,==,!=
- Logical operator:
  - && (and), | | (or)
- Increment and decrement operators:
  - ++,--,++,--
- Assignment operators: +=,-=,/=…



# Relation Operators

- Relation operators has lower precedence than arithmetic operators:
  - >, >=, <, <=
  - ==, !=
- Don’t confuse = and == ! The compiler will warn “suggest parens”.

```
int x=5;
if (x==6)    /* false */
{
  /* ... */
}
/* x is still 5 */
```

```
int x=5;
if (x=6)    /* always true */
{
  /* x is now 6 */
}
/* ... */
```

# Increment and Decrement Operators

| | | | |
|---|---|---|---|
| `x++` | post-increment x | `++x` | pre-increment x |
| `x--` | post-decrement x | `--x` | pre-decrement x |

Note the difference between ++x and x++:

```
int x=5;
int y;
y = ++x;
/* x == 6, y == 6
*/
```

```
int x=5;
int y;
y = x++;
/* x == 6, y == 5
*/
```

# Assignment Operators

- Most binary operators have a corresponding assignment operator *op*
  - +, -, *, /, %, <<, >>, &, ^, |
  - *expr1 op= expr2  <=> expr1 = (expr1) op (expr2)*
  - *x *= y+1 <=> x = x* (y+1)*
- Assignment statement has a value

# Bitwise Operations

- Applied to char, int, short, long
  - And &
  - Or |
  - one's complement ~
  - Exclusive Or ^
  - Left-shift <<
  - Right-shift >>

# Bitwise Operations

<table>
<tr><th>Operator</th><th>Typical Usage</th><th>Express</th></tr>
<tr><td>& (AND)</td><td>1. Take one specific bit</td><td>n & 0x10; ($5^{th}$ bit)</td></tr>
<tr><td></td><td>2. Put several bits to 0 while keep others</td><td>n=n&0xffffff00; (lower 8 bits to 0)</td></tr>
<tr><td>| (OR)</td><td>1. Put several bits to 1 while keep others</td><td>n=n|0xff; (lower 8 bits to 1)</td></tr>
<tr><td>~ (Complement)</td><td>1. Put all bits to complement (not “+ to –”)</td><td>n=~n; (~1 → -2)</td></tr>
<tr><td>^ (XOR)</td><td>1. Put one bit or a set of bits to complement</td><td>n=n^0x10; ($5^{th}$ bit turns to its complement)</td></tr>
<tr><td></td><td>2. Switch two values</td><td>a= a^b; b=a^b; a=a^b;</td></tr>
<tr><td><<</td><td>1. Equivalent to *2^i</td><td>n<<=i;</td></tr>
<tr><td>>></td><td>2. Equivalent to /2^i</td><td>n>>=i;</td></tr>
</table>

# Example: Bit Count

```
/*
   count the 1 bits in a number
   e.g. bitcount(0x45) (01000101 binary) returns 3
*/

int bitcount (unsigned int x) {
   int b;

   for (b=0; x != 0; x = x >> 1)
      if (x & 01)      /* octal 1 = 000000001 */
         b++;

   return b;
}
```

# Operator Precedence and Associativity

highest: + - (unary)
* / %
lowest: + - (binary)

-i * -j = (-i) * (-j)
+i + j / k = (+i) + (j / k)

left associative: it groups from left to right
right associative: it groups from right to left

The binary arithmetic operators (*, /, %, + and -) are all left associative (from left to right)

i – j – k = (i – j) – k    i * j / k = (i * j) / k

The unary arithmetic operators( + and -) are both right associative

\- + i = - ( +i )

# Expression Evaluation

| Precedence | Name | Symbol(s) | Associativity<br>结合关系 |
|---|---|---|---|
| 1 | X++/X-- | | Left<br>(from left to right) |
| 2 | ++X/--X<br>unary +/-(单目) | | Right<br>(from right to left) |
| 3 | multiplicative | *, /, % | left |
| 4 | additive | +, - | left |
| 5 | logical | &&,\|\| | left |
| 6 | assignment | =, *=, /=, +=, -= | Right |

# Expression Evaluation examples

```
int a,b,c,d,e,f;
    a=1;    b=2;     c=3;
    d = !(a+b)+c-1&&b+c/2;
    e = a--&&b++&&++c;
    f = a+=b++-c*d ;
```

# Expression Evaluation examples

```
int a,b,c,d,e,f,g;
    a=1;    b=2;     c=3;
    d = !(a+b)+c-1&&b+c/2;
    e = a--&&b++&&++c;
    f = a+=b++-c*d ;
```

a: -1
b: 4
c: 4
d: 1
e: 1
f: -1

# Conditional Expressions

- Conditional expressions
- expr1? expr2:expr3;
- if expr1 is true then expr2 else expr3

```
(u%2==0)?printf(“even num\n”):printf(“odd num\n”);
```

# Agenda

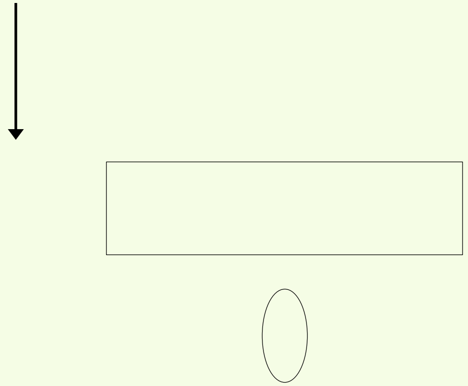
- Data type and storage
- Expression
- Control flow

# Control Structures

- Sequence execution
  - Statements executed one after the other in the order written
- Selection structures:
  - if, if/else, and switch
- Repetition (Loops) structures:
  - while, for and do/while
- Transfer controls:
  - goto, breaks, continues

# Control Structures

- Flowchart
  - Graphical representation of an algorithm
  - Drawn using certain special-purpose symbols connected by arrows called *flowlines.*
  - Rectangle symbol (action symbol): indicates any type of action.
  - Oval symbol: indicates beginning or end of a program, or a section of code (circles).



- Single-entry/single-exit control structures
  - Connect exit point of one control structure to entry point of the next (*control-structure stacking*).
  - Makes programs easy to build

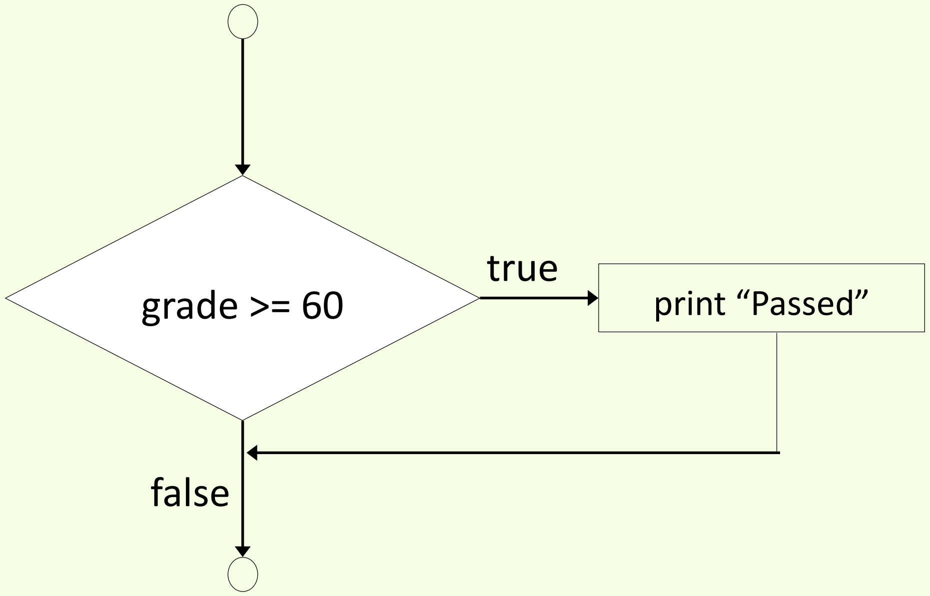
# The `if` Selection Structure

- Selection structure:
  - Used to choose among alternative courses of action
  - Pseudocode: *If student's grade is greater than or equal to 60*
    *Print "Passed"*
- If condition **`true`**
  - Print statement executed and program goes on to next statement.
  - If **`false`**, print statement is ignored and the program goes onto the next statement.
  - Indenting makes programs easier to read
    - C ignores whitespace characters.
- Pseudocode statement in C:

```
if ( grade >= 60 )
   printf( "Passed\n" );
```

# The `if` Selection Structure (II)

- Diamond symbol (decision symbol) - indicates decision is to be made
  - Contains an expression that can be **`true`** or **`false`**
  - Test the condition, follow appropriate path
- **`if`** structure is a single-entry/single-exit structure.




A decision can be made on any expression.
zero - **`false`**
nonzero - **`true`**
Example:
**`(3 – 4)`** is **`true`**

# The `if/else` Selection Structure

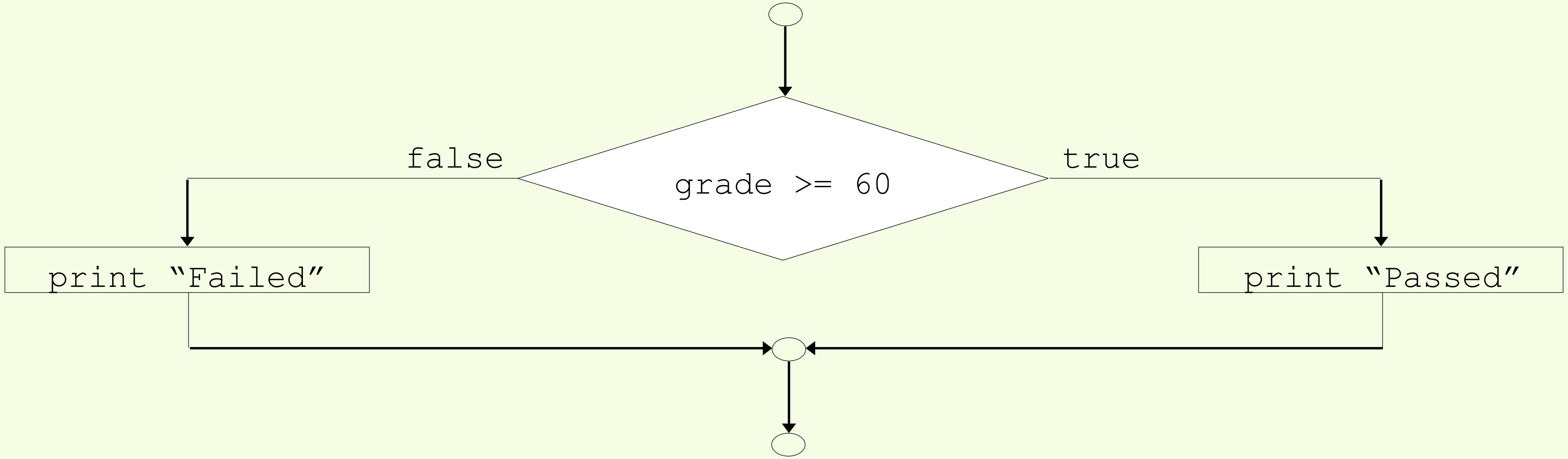
- `if`
  - Only performs an action if the condition is **`true`**.
- `if/else`
  - A different action when condition is **`true`** than when condition is **`false`**
- Psuedocode: *If student's grade is greater than or equal to 60*
  *Print "Passed"*
  *else*
  *Print "Failed"*
  - Note spacing/indentation conventions
- C code:

```
if ( grade >= 60 )
   printf( "Passed\n");
else
   printf( "Failed\n");
```

or

```
if ( grade >= 60 ){
  printf( "Passed\n");
}
else{
  printf( "Failed\n");
}
```

# The `if/else` Selection Structure (II)




- Ternary conditional operator (**`?:`**)
  - Takes three arguments (condition, value if **`true`**, value if **`false`**)
  - Our pseudocode could be written:

```
printf( "%s\n", grade >= 60 ? "Passed" : "Failed" );
```

OR

```
grade >= 60 ? printf( "Passed\n" ) : printf( "Failed\n" );
```

# The `if/else` Selection Structure (III)

- Compound statement:
  - Set of statements within a pair of braces
  - Example:

```
if ( grade >= 60 )
   printf( "Passed.\n" );
else {
   printf( "Failed.\n" );
   printf( "You must take this course again.\n" );
}
```

  - Without the braces,

```
printf( "You must take this course again.\n" );
```

  would be automatically executed
- Block: compound statements with declarations

# The `if/else` Selection Structure (IV)

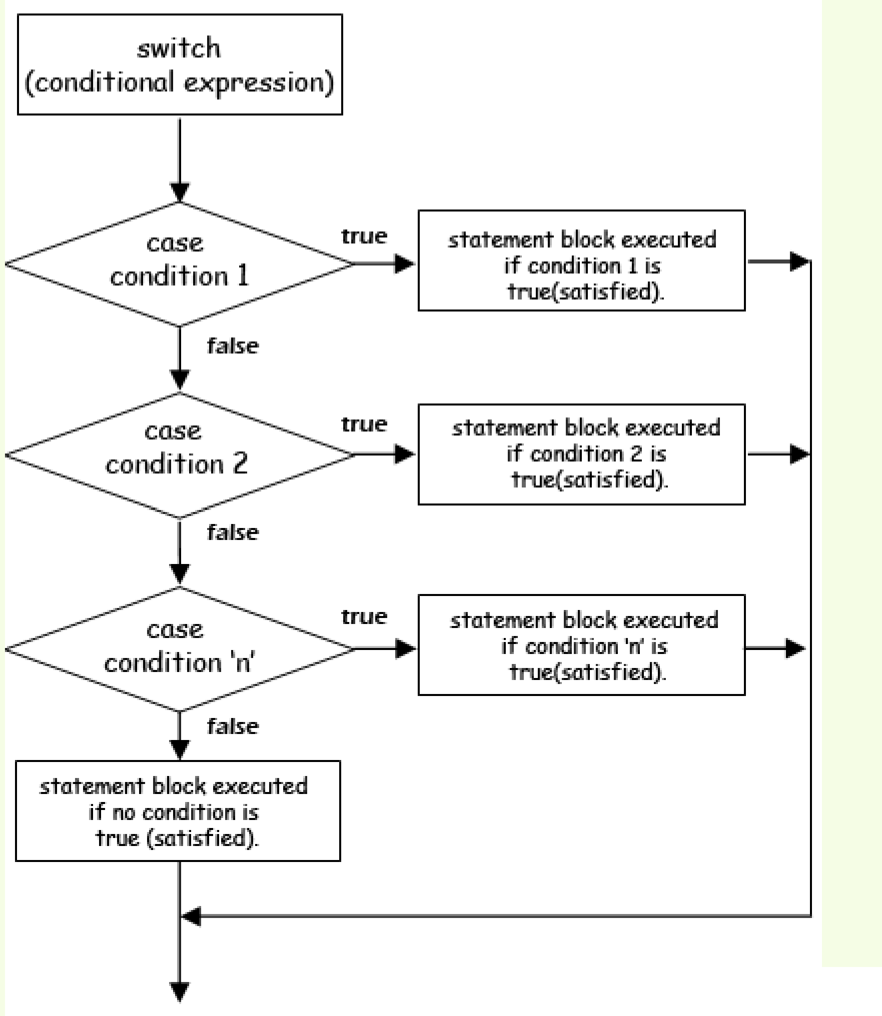
- Nested **`if/else`** structures
  - Test for multiple cases by placing **`if/else`** selection structures inside **`if/else`** selection structures

*If student's grade is greater than or equal to 90*
    *Print "A"*
*else*
    *If student's grade is greater than or equal to 80*
        *Print "B"*
    *else*
        *If student's grade is greater than or equal to 70*
            *Print "C"*
        *else*
            *If student's grade is greater than or equal to 60*
                *Print "D"*
            *else*
                *Print "F"*

  - Once condition is met, rest of statements skipped
  - Deep indentation usually not used in practice

# The `Switch` Selection Structure

- The `switch` statement is a multi-way decision that tests whether an expression matches one of a number of **constant integer** values



# The `Switch` Selection Structure (II)

```
char grade;
float GPA=0;
scanf("%c",&grade);
switch (grade) {
    case "A":     GPA+=4.0;
    break;
    case "B":     GPA+=3.5;
    break;
    default:  GPA+=2.7;
    }
```

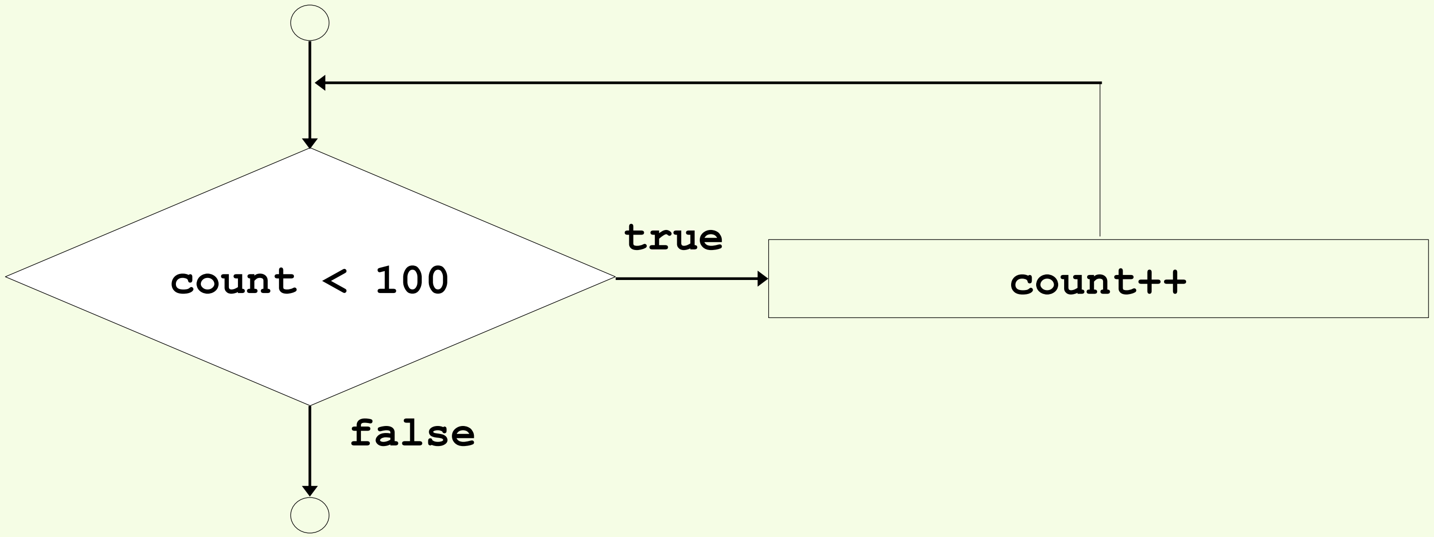
case 'A'

- The `break` statement causes an immediate exit from the switch.

# The `while` Repetition Structure (II)

- Repetition structure
  - Programmer to specify an action to be repeated while some condition remains **`true`**
- Example:

```
int count = 1;
while ( count < 100 )
    count++;
```



# The `while` Repetition Structure

- Repetition structure
  - Programmer to specify an action to be repeated while some condition remains **`true`**
  - Psuedocode: *While there are more items on my shopping list*

    *Purchase next item and cross it off my list*
  - **`while`** loop repeated until condition becomes **`false`**

```
int count = 0;
while ( count < 100 )
   count++;
```

When stopped,
count =?

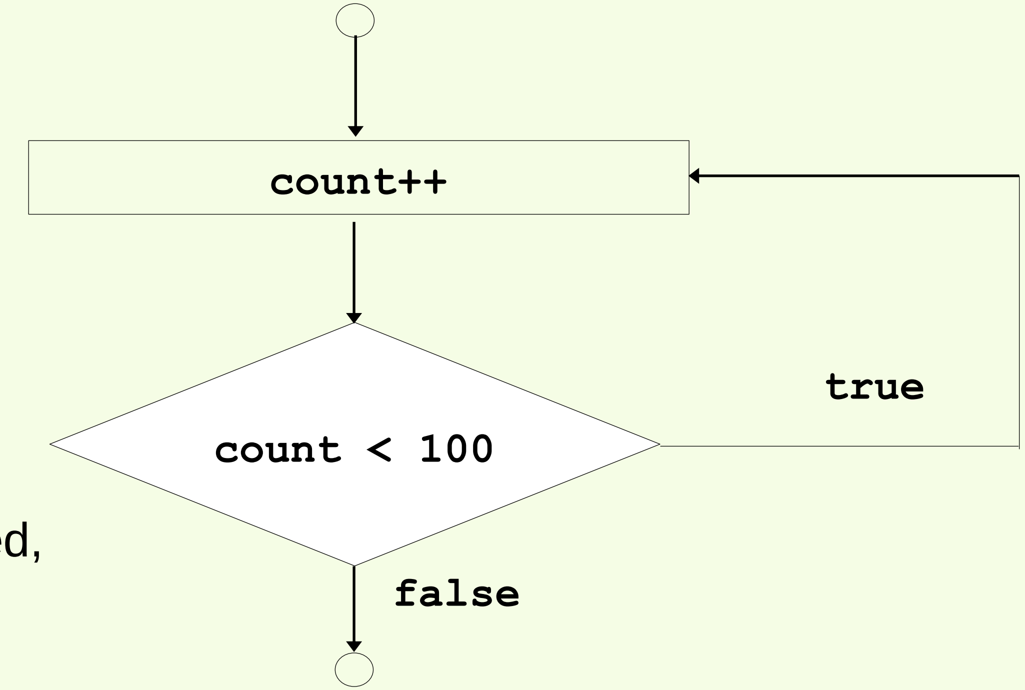
# The Do Repetition Structure

- Repetition, statement will be excuted once before do the judgement

```
int count=1;
do {
     count++;
} while (count<100);
```

```
do{
   statement(s);

}while( condition );
```




When stopped,
count =?

# Exercise 1

- Input 4 3, output? count =?
- Input 3 5, output? count =?
- What does this code do?

```
#include <stdio.h>

int main()
{
    int a,n,count=1;
    long int sn=0, tn=0;
    scanf("%d %d", &a, &n);
    while(count<=n)
    {
        tn=tn+a;
        sn=sn+tn;
        a=a*10;
        count++;
    }
    printf("%ld\n",sn);
}
```

# Exercise 2

- Narcissistic number
  - For a number n in base b>1, it's has k digitals, We define a narcissistic function Fb(n): $\mathbb{N} \to \mathbb{N}$

$$F_b(n) = \sum_{i=0}^{k-1} d_i^k$$

  - N is a narcissistic number if $F_b(n) = n$

三位的水仙花数

四位的四叶玫瑰数

五位的五角星数

六位的六合数

七位的北斗七星数

八位的八仙数

九位的九九重阳数

# Data Type and storage

- Integer:
  - Two's complement encoding
  - Boolean
- float, double, long double
  - **IEEE 754 float: $N = (-1)^S \times 1.M \times 2^{E-127}$**

| S | E | M |
| --- | --- | --- |
| 1 bit | 8 bits | 23bits |

  - Not precise! Don't do == with float
- Char: 1 Byte

# Expression

- Operators
- Associativity
  - –Left, right
- Precedence
- Evaluation of expression