

# Lecture 1: C Programming (Part I)

---

01204212 Abstract Data Types and Problem Solving

Department of Computer Engineering  
Faculty of Engineering, Kasetsart University  
Bangkok, Thailand.



Department of  
**Computer Engineering**  
Kasetsart University



# Outline

---

- Programming Overview
- Statement
- Expression
- Variable
- Data Types
- Selection Control Structure

# A computer program

- A **computer program** is a sequence of **instructions** to be executed by computers.
- Examples of computer programs in various forms:

```
0001 1001  
1001 1110  
1000 1011  
1100 1011  
1110 0010  
1001 0111  
1100 1011  
1110 0010  
1001 0111  
1100 1011
```

```
MOV AX,10  
SUB BX,AX  
MOV [DX],AX  
JMP 200  
MOV CX,5  
MOV AX,10  
MUL AX,CX  
CMP BX,AX  
JLE 500  
JMP 400
```

Machine  
instructions

Instructions in  
assembly language

```
int i, sum = 0;  
for (i=0; i<100; i++)  
    sum += i;
```

Instructions in C language

More readable

# High-Level Language

---

- Easy to learn, understand, write program
  - Program instructions are English-like statements
- Safeguards against bugs
  - Rules and conditions are enforced at compile-time or run-time
- Built-in library functions
  - Many built-in functions are provided to perform specific tasks of new programs
- Machine independence
  - Operations do not depend on instruction set
    - e.g., can write “ $a = b * c$ ”, even though LC-3 processor does not have a multiply instruction
  - A program written in one type of computer can be executed on another type of computer

# Programming Paradigms

---

- Procedural Programming
  - Paradigm that uses a list of instructions to tell the computer **what to do step-by-step** (i.e., top-down approach)
  - It relies on **procedures**, a.k.a. subroutines, containing a series of computational steps to be carried out
  - Languages: **C**, Fortran, Cobol, Pascal, ...
- Object-Oriented Programming
  - Approach to problem-solving where all computations are carried out using **objects**
  - Object is a component of a program that knows how to perform certain **actions** and how to interact with other **elements** of the program
  - Languages: C#, Java, ...

# Interpretation vs. Compilation

---

- Different ways of translating high-level language
- Interpretation
  - Convert program into machine code when the program is run
  - Execute **each statement** at a time
  - Tool: **interpreter**
  - Languages: Perl, Bash, PHP, Python, JavaScript, ...
- Compilation
  - Convert **the whole program** into machine code before running
  - Perform **optimization** over multiple statements
  - Tool: **compiler**
  - Languages: **C**, C++, C#, ...

# Interpretation vs. Compilation

---

- Different ways of translating high-level language
- Interpretation

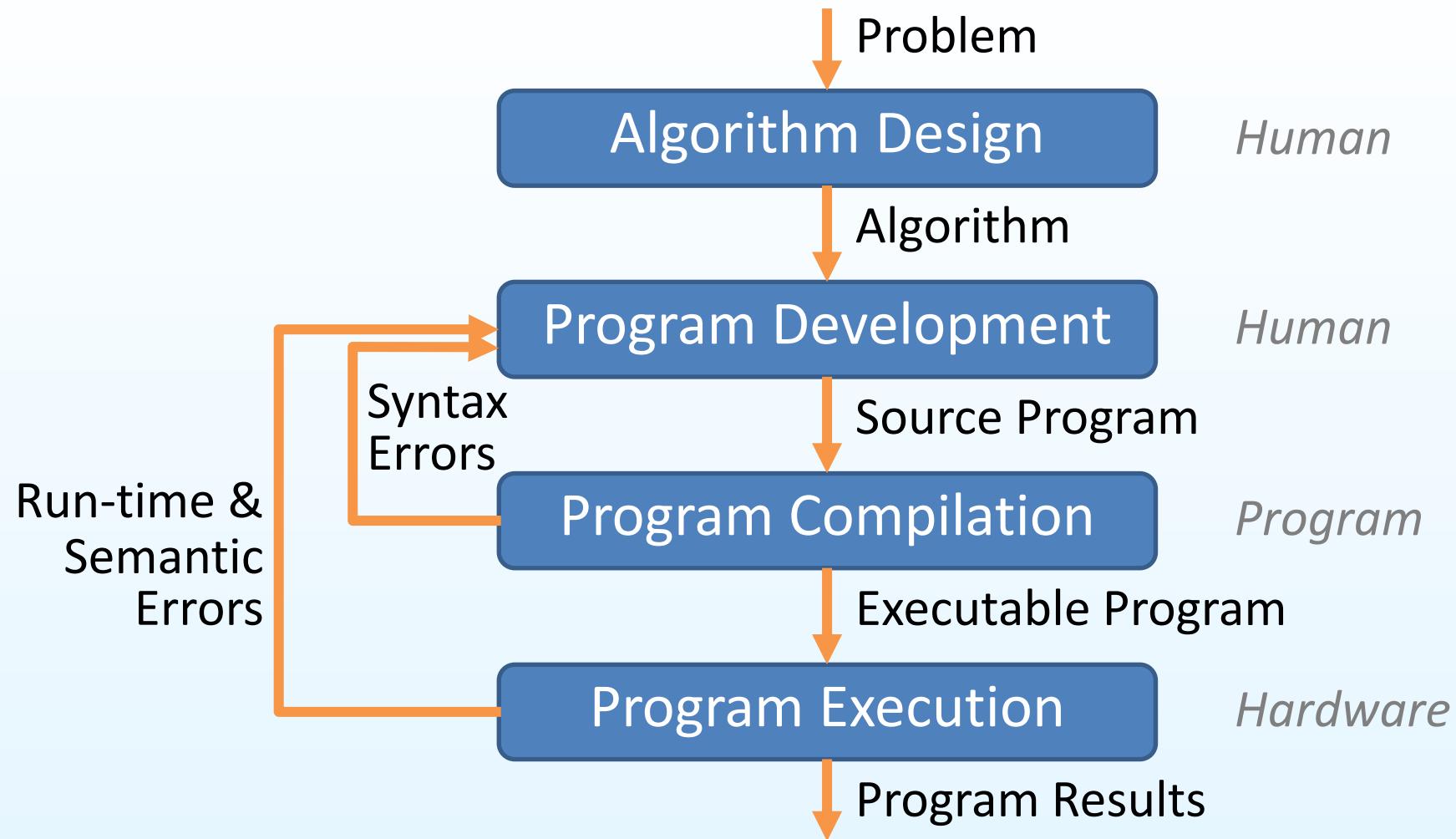
```
repeat 100 times
    n = 10
    sum = sum + n
print sum
```

- Compilation

```
n = 10
repeat 100 times
    sum = sum + n
print sum
```

What are differences?

# Development Process



# C Compilation Process

## Preprocessor

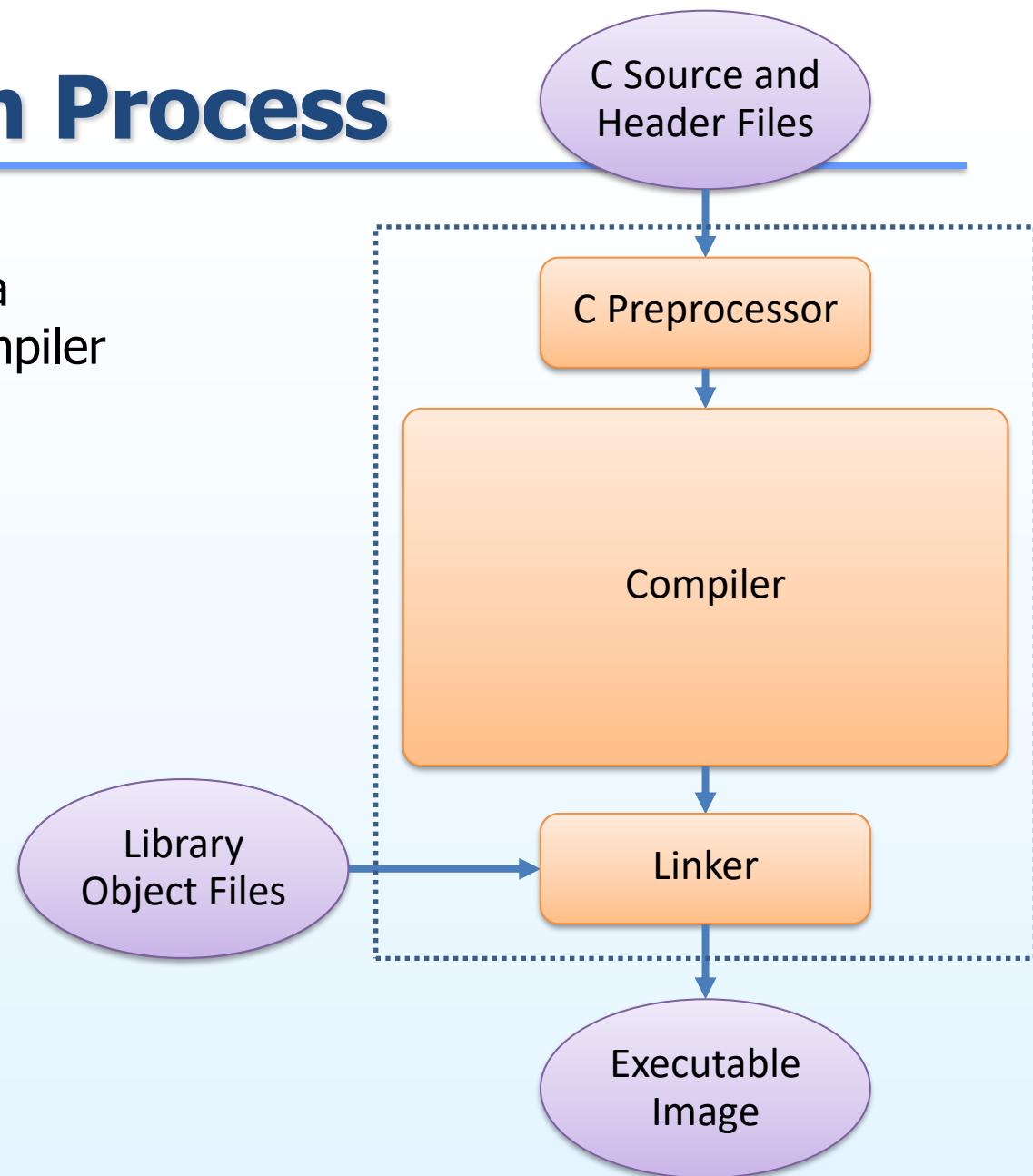
- Expand source code to a simpler form for the compiler
- Expand macros
- Strip out comments
  - ➔ Still C code

## Compiler

- Generate object file
  - ➔ Machine instructions

## Linker

- Combine object files (including libraries)
  - ➔ Executable image



# C Compilation Process

## Source Code Analysis

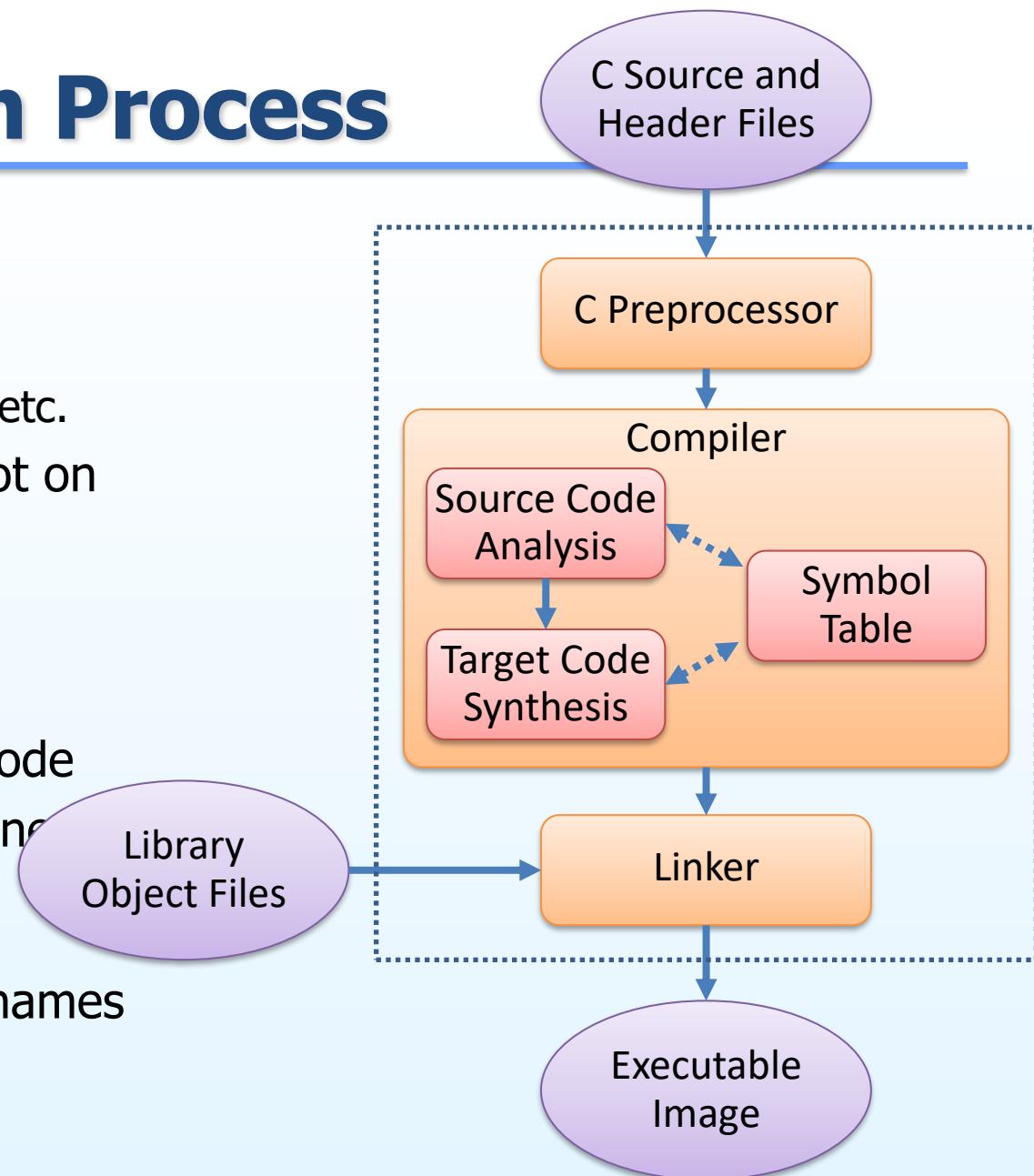
- Parse program to pieces
  - Variables, expressions, statements, functions, etc.
- Depend on language (not on target machine)

## Target Code Synthesis

- Generate machine code
- May optimize machine code
- Depend on target machine

## Symbol Table

- Map between symbolic names and items



# First C Program with “Hello World”

```
1: #include <stdio.h>
2: /* A simple C program */
3: int main(int argc, char **argv)
4: {
5:     printf("Hello World\n");
6:     return 0;
7: }
```

A **header** file: Interface to libraries and other C files

A **comment**: The compiler ignores this

The **main()** function: Your program starts here

A **Block of code**: The scope is marked by { ... }

A **printf()** function: Print out a message

A **return** statement: Return a value from this function

# A Quick Digression about Preprocessor

```
#include <stdio.h>
/* A simple C program */
int main(int argc, char **argv)
{
    printf("Hello World\n");
    return 0;
}
```

## Preprocessing

```
_extension_ typedef unsigned long long int __dev_t;
_extension_ typedef unsigned int __uid_t;
_extension_ typedef unsigned int __gid_t;
_extension_ typedef unsigned long int __ino_t;
_extension_ typedef unsigned long long int __ino64_t;
_extension_ typedef unsigned int __nlink_t;
_extension_ typedef long int __off_t;
_extension_ typedef long long int __off64_t;
extern void flockfile (FILE *_stream);
extern int ftrylockfile (FILE *_stream);
extern void funlockfile (FILE *_stream);
int main(int argc, char **argv)
{
    printf("Hello World\n");
    return 0;
}
```

Compiling  
and Linking

Executable  
File “a.out”

### In preprocessing step,

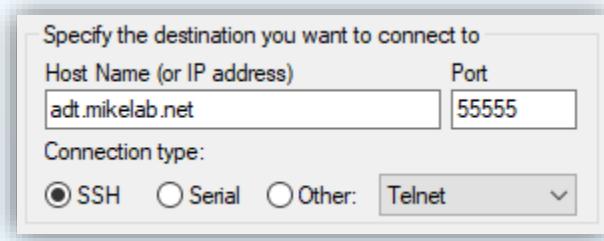
- Parse included files (`#include`)
- Expand macros (`#define`)
- Strip out comments (`/* */`, `//`)
- Join continued lines (`\`)

### In compiling step,

- Convert to binary code

# The ADT Class Environment

- Server: **adt.mikelab.net**
- User: **b<your 10-digit student ID>**
- Pass: ... Let you know in the class
- For Windows, use the **Putty** program



- For MAC or Linux, use the **terminal** application

```
$ ssh <user>@adt.mikelab.net -p 55555
```

# The ADT Class Environment

Cannot be used

- Use the **vi** Editor ... see [tutorial](#)
- Special commands for your assignment tasks
  - Show the status of your work



```
$ hw-status [--all]
```

- Submit your job

```
$ hw-send LABNAME FILENAME
```

- Retrieve your job that have already been submitted

```
$ hw-load LABNAME
```

# Writing and Running Program

1. Write a C program, named “hello.c”

```
#include <stdio.h>
/* A simple C program */
int main(int argc, char **argv)
{
    printf("Hello World\n");
    return 0;
}
```

2. Convert the program to an executable file

```
$ gcc hello.c
```

option:  
-o specify the executable name  
-Wall enables all the warnings  
-c produce only the compiled code  
-E produce only the preprocessor output  
-S produce only the assemble code  
-save-temp produce all the intermediate files  
-l link with shared libraries

3. Run the program

```
$ ./a.out
Hello World
$
```

# Computer Memory

---

- The smallest unit of information storage on a computer, called **bit**

- Can be either **0** or **1**



- A group of eight bits, called **byte**

- Store a value ranging from **0** to **255**



# Computer Memory

- Memory is like a big table of slots where bytes can be stored
- 1-byte **value** can be stored in each slot, referred by its **address**
- Some logical data values span more than one slot:
  - E.g., the character string "Hello\n"
- A **data type** names a logical meaning to a span of memory:
  - char** a single character (1 byte)
  - int** signed integer (4 bytes)
  - float** floating point (4 bytes)
  - double** double-precision floating point (8 bytes)

Address	Value
high ...	
0x7fffb5	1 byte
0x7fffb4	00000000 (0 = '\0')
0x7fffb3	00001010 (10 = '\n')
0x7fffb2	01101111 (111 = 'o')
0x7fffb1	01101100 (108 = 'l')
0x7fffb0	01101100 (108 = 'l')
0x7ffffaf	01100101 (101 = 'e')
0x7ffffae	01001000 (72 = 'H')
0x7ffffad	
0x7ffffac	
0x7ffffab	
0x7ffffaa	
0x7ffffa9	
0x7ffffa8	
low ...	

# Statement

---

- A statement is a **complete command** to ask the computer to do something
- Can be a single-line command, ending with a semicolon ;

```
printf("Hello World\n");
```

or

```
return 0;
```

- Can be a block of codes, marked by the braces { ... }

```
for (i=0; i<10; i++)
{
    printf("Hello World\n");
    sleep(1);
}
```

# Output Statement: printf() Function

- The built-in I/O functions are provided in C standard library “`stdio.h`”
- Here, we use the `printf()` function to **formatted output** to **standard output**

Syntax:

```
printf(ctrl_string, var_list);
```

- Examples:

```
printf("This is simple text for output\n");
printf("I am %d years old\n", age);
printf("The date is %3d%3d%6d\n", 2, 5, 2003);
printf("The answer is %5.2f\n", someFloatValue);
```

# Identifier

---

- A name for some C elements, such as variables or functions
- Some common rules
  - Consist of only alphabet, digits, and underscores
  - Cannot begin with a digit
  - Must not be a reserved word
  - Be case sensitive

# Reserved Words (Keywords)

---

Keywords			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

# Variable

---

- A **variable** names a **place in memory** where you store a value of a certain type
- You first **define** a variable before using by giving it a **name** and specifying the **type**, and optionally an **initial value**

Syntax:

```
datatype variable;  
datatype variable = value;
```

# C Data Types

- A data type determines the **type** and **size** of data associated with a variable

Data Type	Size (bytes)	Format Specifier (placeholder)
char	1	%c
short	2	%hd, %hi, %d, %i
int	4	%d, %i
long	8	%ld, %li
float	4	%f
double	8	%lf, %f

- 1) Try to use **%d** for **char**!
- 2) Explore other data types by yourself

# OK, We're Back ... Variable Declaration

```
char x = 'e';
char y;
int sum;
```

What are the  
values of y  
and sum?

Symbol	Address	Value
	...	
	0x7fffb2	
	0x7fffb1	????????
	0x7fffb0	????????
	0x7ffffaf	????????
sum	0x7ffffae	????????
	0x7ffffad	
	0x7ffffac	
x	0x7ffffab	01100101 (101 = 'e')
y	0x7ffffaa	????????
	0x7ffffa9	
	0x7ffffa8	
	...	

# Input Statement: `scanf()` Function

- The built-in I/O functions are provided in C standard library “`stdio.h`”
- Here, we use the `scanf()` function to **formatted input** from **standard input**

Syntax:

```
scanf(ctrl_string, var_list);
```

- Examples:

```
scanf("%d %d", &width, &height);
scanf("%c%c%c", &a, &b, &c);
scanf("I am %d years old", &age);
```

# Arithmetic Expression

---

- An expression is something that can be evaluated to a **type** and a **value**
- An arithmetic expression can be evaluated to a **numerical value**
- Examples:
  - 1
  - 1+2
  - (1+2)/3

# Some C Operators

---

- Arithmetic operators
  - Unary: plus(+), minus(-)
  - Binary: addition(+), subtraction(-), multiplication(\*), division(/), modulus(%)
- Assignment operators
  - Assignment: =
  - Shortcut assignment: +=, -=, \*=, /=, %=
- Increment and decrement operators
  - Unary prefix: `++var`, `--var`
  - Unary postfix: `var++`, `var--`

What is the  
value of  $1/2$ ?

# Precedence and Associativity

high  
↓  
low

Type	Operator	Associativity
Unary postfix	<code>++ --</code>	left to right
Parentheses	<code>( )</code>	left to right
Unary prefix	<code>++ -- + -</code>	right to left
Multiplicative	<code>* / %</code>	left to right
Additive	<code>+ -</code>	left to right
Assignment	<code>= += -= *= /= %=</code>	right to left

## Quiz:

```
1: #include <stdio.h>
2: int main(void)
3: {
4:     int x = 2, y = 5;
5:     x += ++y;
6:     printf("%d %d\n", x, y);
7:     x += y++;
8:     printf("%d %d\n", x, y);
9:     return 0;
10: }
```

What is the output  
of this program?

# Data Type Conversion

---

```
int      x = 10;  
double  y = 20.5;
```

- Arithmetic promotion (implicit)
  - Operators convert their operand automatically
$$y = y + x;$$
- Casting (explicit)
  - Conversion is performed by the programmer via an explicit type
$$x = (\text{int})y;$$

# Exercise 1 (5 mins.)

---

Write a program that asks the user to enter the radius of a circle and then computes and displays the circle's area.

Use the formula:

$$Area = PI \times Radius \times Radius$$

where *PI* is the constant macro 3.14159.



# Fundamental Flow Controls

---

- Sequence
  - Subroutine (function)
- Selection: if, if-else, switch
- Repetition: for, while, do-while

# More C Expressions and Operators

- A Boolean expression can be evaluated to a Boolean result
  - **NOTE:** In C program, 0 refers to False and 1 refers to True
- Boolean operators
  - Relational: ==, !=, >, <, >=, <=
  - Logical: NOT(!), AND(&&), OR(||)
  - Bitwise: NOT(~), AND(&), XOR(^), OR(|)

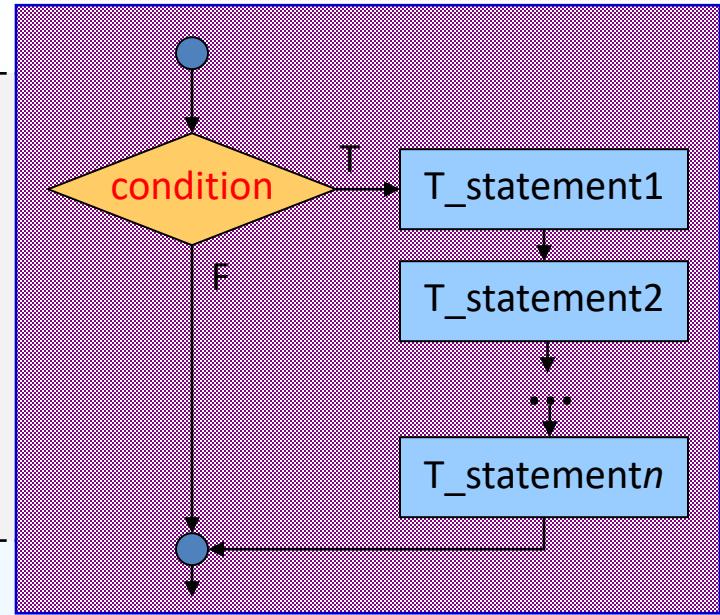
Also, non-zero value !!!

What is the value  
of the expression  
**! -2<3&&3<2?**

# Selection: The if Statement

Syntax:

```
if (condition)
{
    T_statement1;
    T_statement2;
    ...
    T_statmentn;
}
```

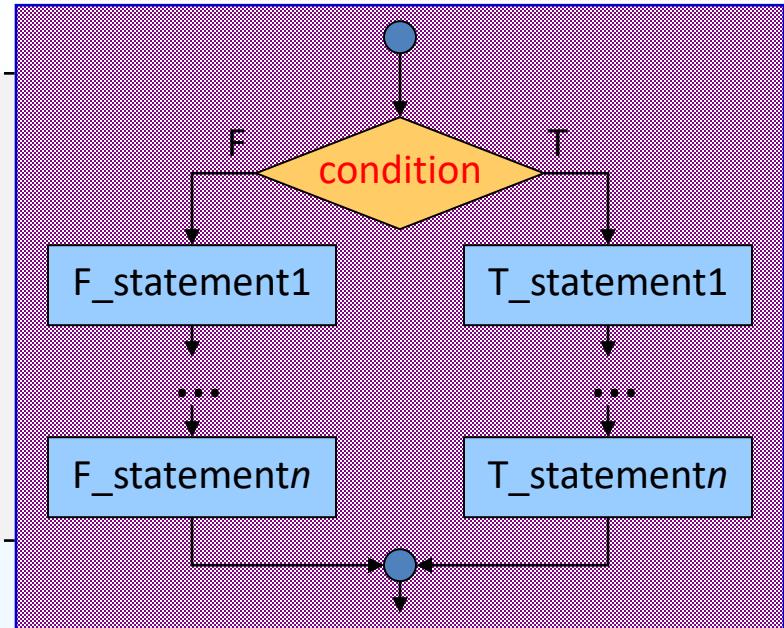


- Use the **if** keyword
- The statements are **performed** if the **condition** is **True** (non-zero value), otherwise they are skipped
- The braces **{ }** are used to determine a **block**
  - Can be omitted if there is only one statement to be performed

# Selection: The if-else Statement

Syntax:

```
if (condition) {  
    T_statement1;  
    ...  
} else {  
    F_statement1;  
    ...  
}
```



- Use the **if** and **else** keywords
- Specify which statements to be **performed** when the **condition** is either **True** or **False**
- The braces **{ }** are used to determine a **block**
  - Can be omitted if there is only one statement to be performed

## Exercise 2 (5 mins.)

---

Write an interactive program that contains an **if** statement that computes the area of either **a square** ( $area = side^2$ ) or **a triangle** ( $area = 1/2 * base * height$ ) after prompting the user to type the **first character** of the figure name (**S** or **T**).



# Selection: The ternary Operator

---

- A ternary operator is a basic conditional expression

Syntax:

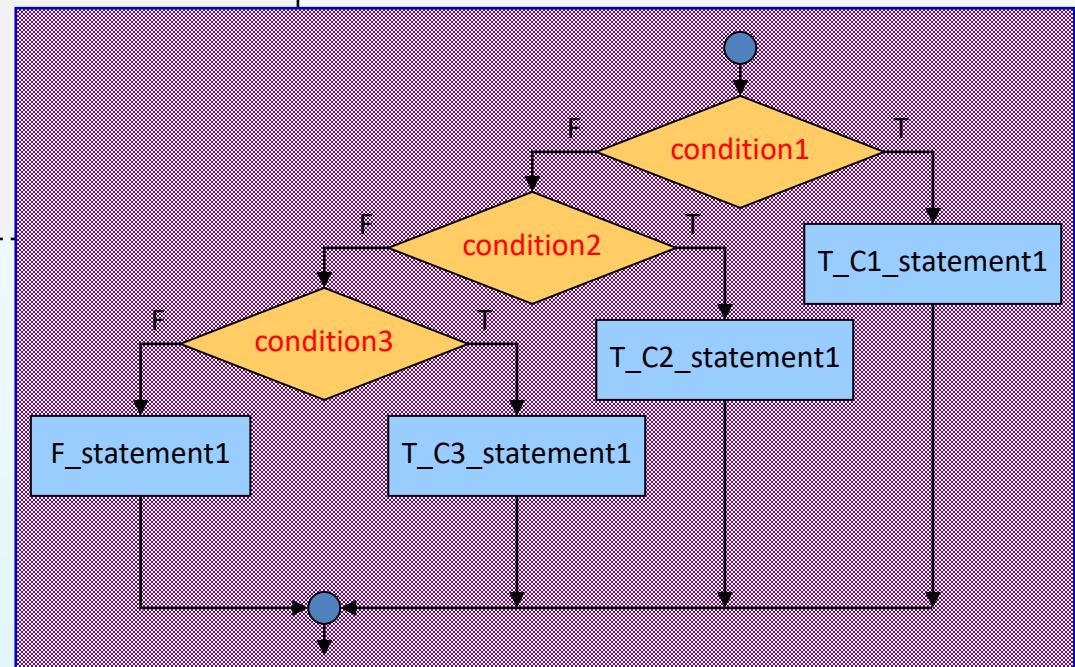
*condition* ? *exp1* : *exp2*;

- The expression will evaluate to *exp1* if the **condition** is **True**, and otherwise to *exp2*
- Examples:  
`absdiff = y > x ? y - x : x - y;`  
`grade >= 50 ? printf("Passed\n") : printf("Failed\n");`

# Nested Selection

Syntax:

```
if (condition1) {  
    T_C1_statement1;  
} else if (condition2) {  
    T_C2_statement1;  
} else if (condition3) {  
    T_C3_statement1;  
} else {  
    F_statement1;  
}
```



# Examples: Nested Selection

Code Fragment #1:

```
1: if ((letter >= 'A') && (letter <= 'Z'))
2:   if (number >= 10)
3:     printf("*****\n");
4:   else
5:     printf("#####\n");
```

Code Fragment #2:

```
1: if ((letter >= 'A') && (letter <= 'Z'))
2:   if (number >= 10)
3:     printf("*****\n");
4:   else
5:     printf("#####\n");
```

What is the output if

- 1) letter is 'B' and number is 3
- 2) letter is 'b' and number is 3

# Exercise 3 (5 mins.)

---

Write a program to get the first character from user and then display the class of ships as the following decision table:

Class ID	Ship Class
B or b	Battleship
C or c	Cruiser
D or d	Destroyer
F or f	Frigate

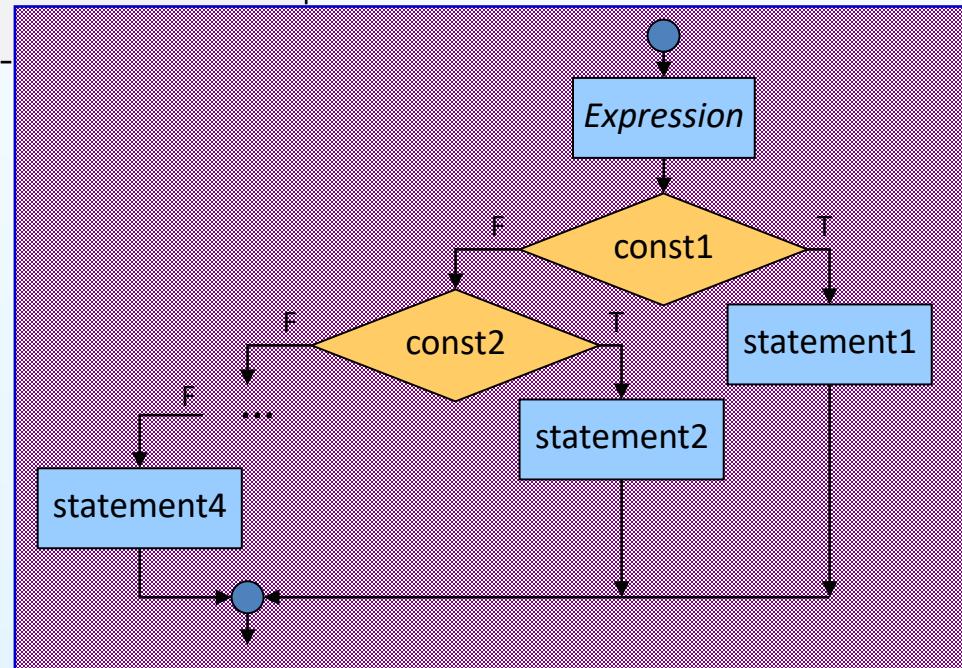


# Selection: The `switch` Statement

## Syntax:

```
switch (expression) {  
    case const1: statement1; break;  
    case const2: statement2; break;  
    ...  
    default: statementn; break;  
}
```

- Use the `switch`, `case` and `default` keywords
- Group of the selected statements is performed, **terminated** by a `break`
- A valid expression may be of type `int`, `char`, `enum`.



# Examples: The switch Statement

Code Fragment #1:

```
1: switch (count) {  
2:     case 0: printf("zero\n"); break;  
3:     case 1: printf("one\n"); break;  
4:     case 2: printf("two\n"); break;  
5:     default: printf("more than two\n"); break;  
6: }
```

What is the

Code Fragment #2:

```
1: switch (count) {  
2:     case 0: printf("zero\n");  
3:     case 1: printf("one\n");  
4:     case 2: printf("two\n");  
5:     default: printf("more than two\n");  
6: }
```

difference between  
two code fragments?

# Again, Exercise 3 (5 mins.)

Write a program to get the first character from user and then display the class of ships as the following decision table:

Class ID	Ship Class
B or b	Battleship
C or c	Cruiser
D or d	Destroyer
F or f	Frigate

Using the switch  
statement



# Any Question?

# Solution to Exercise 1

```
1: #include <stdio.h>
2: #define PI 3.14159
3:
4: int main(void) {
5:     double radius = 0;
6:     double area = 0;
7:
8:     printf("Enter the radius: ");
9:     scanf("%lf", &radius);
10:    area = PI * radius * radius;
11:    printf("The circle's area is %lf\n", area);
12:
13:    return 0;
14: }
```

```
Enter the radius: 10
The circle's area is 314.159000
```



# Solution to Exercise 2

```
1: #include <stdio.h>
2:
3: int main(void) {
4:     char figure = '\0';
5:     double side = 0, base = 0, height = 0, area = 0;
6:
7:     printf("Enter the figure (S or T): ");
8:     scanf("%c", &figure);
9:     if (figure == 'S') {
10:         printf("Enter the side: ");
11:         scanf("%lf", &side);
12:         area = side * side;
13:     }
14:     if (figure == 'T') {
15:         printf("Enter the base and height: ");
16:         scanf("%lf %lf", &base, &height);
17:         area = 1.0/2 * base * height;
18:     }
19:     printf("The area is %lf\n", area);
20:     return 0;
21: }
```

Enter the figure (S or T): S

Enter the side: 10

The area is 100.000000

Enter the figure (S or T): T

Enter the base and height: 10 20

The area is 100.000000



# Solution to Exercise 3

```
1: #include <stdio.h>
2:
3: int main(void) {
4:     char class = '\0';
5:
6:     printf("Enter the class ID: ");
7:     scanf("%c", &class);
8:     if ((class == 'B') || (class == 'b'))
9:         printf("Battleship\n");
10:    else if ((class == 'C') || (class == 'c'))
11:        printf("Cruiser\n");
12:    else if ((class == 'D') || (class == 'd'))
13:        printf("Destroyer\n");
14:    else if ((class == 'F') || (class == 'f'))
15:        printf("Frigate\n");
16:    else
17:        printf("Unknown ship class %c\n", class);
18:    return 0;
19: }
```

Enter the class ID: **d**  
Destroyer



# Solution to Exercise 3 (switch)

```
1: #include <stdio.h>
2:
3: int main(void) {
4:     char class = '\0';
5:
6:     printf("Enter the class ID: ");
7:     scanf("%c", &class);
8:     switch (class) {
9:         case 'B':
10:            case 'b': printf("Battleship\n"); break;
11:            case 'C':
12:            case 'c': printf("Cruiser\n"); break;
13:            case 'D':
14:            case 'd': printf("Destroyer\n"); break;
15:            case 'F':
16:            case 'f': printf("Frigate\n"); break;
17:        default: printf("Unknown ship class %c\n", class);
18:    }
19:    return 0;
20: }
```

Enter the class ID: **d**  
Destroyer

