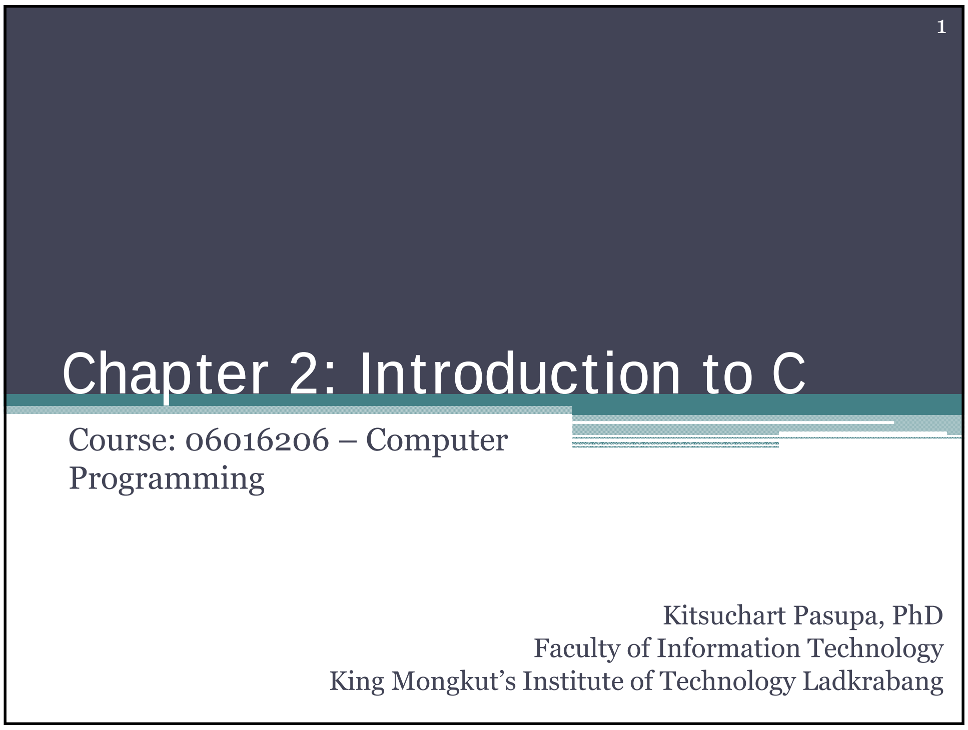
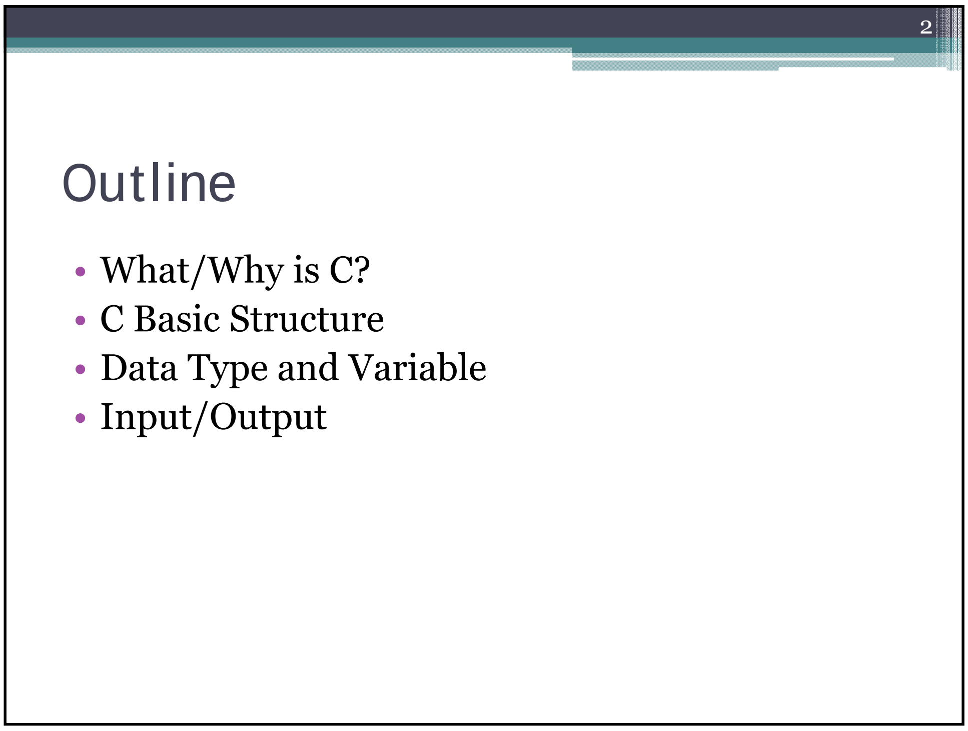


Chapter 2: Introduction to C

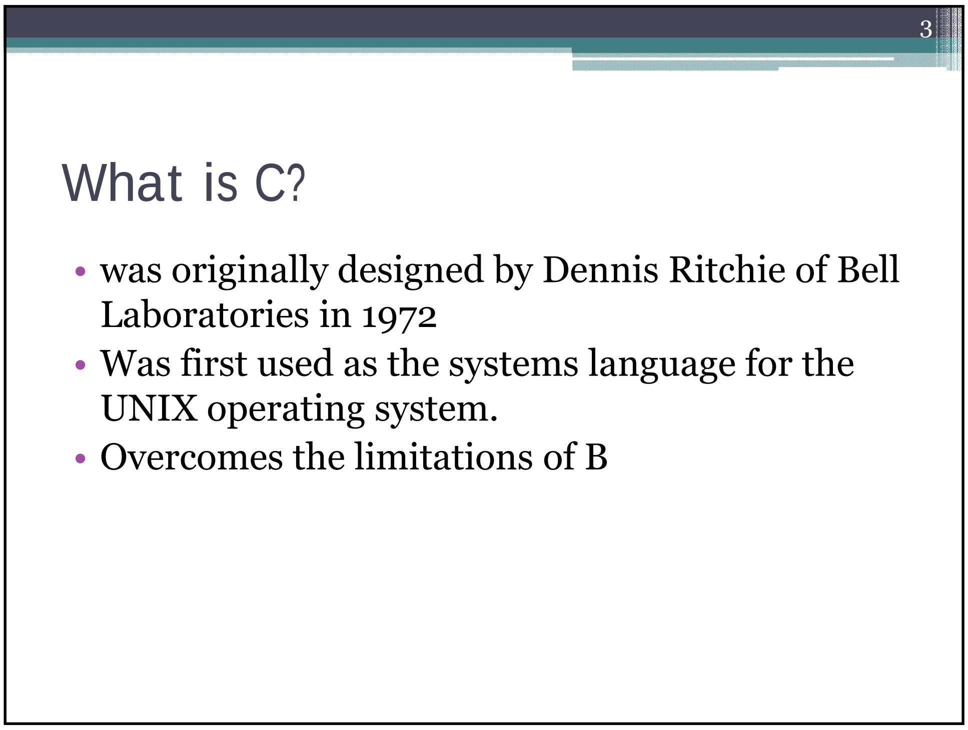
Course: 06016206 – Computer Programming

Kitsuchart Pasupa, PhD
Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang



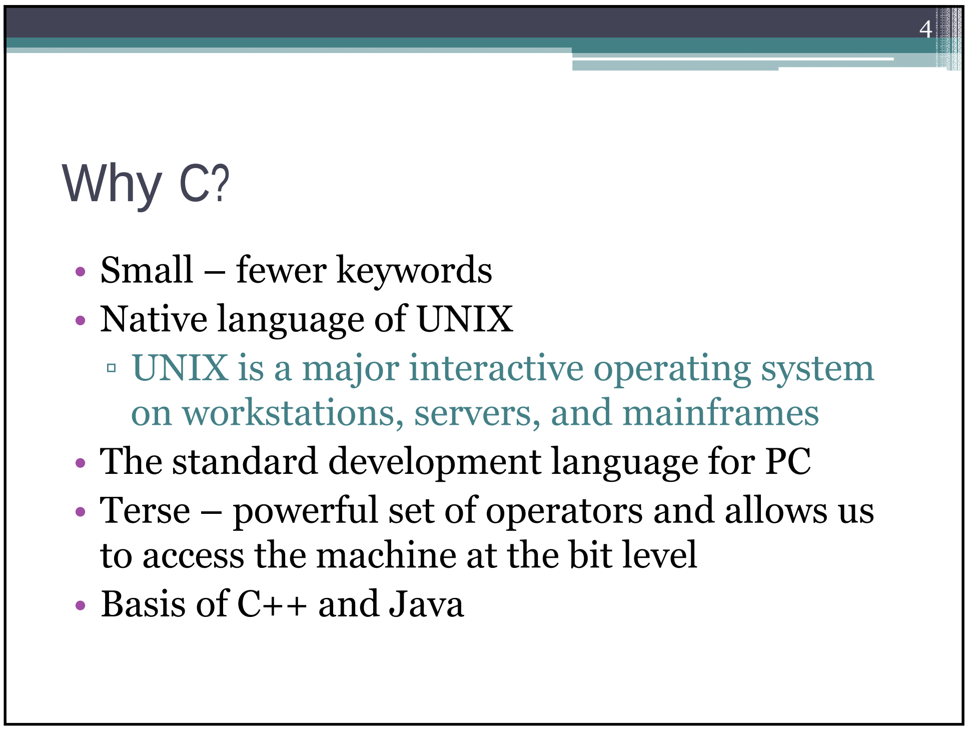
Outline

- What/Why is C?
- C Basic Structure
- Data Type and Variable
- Input/Output



What is C?

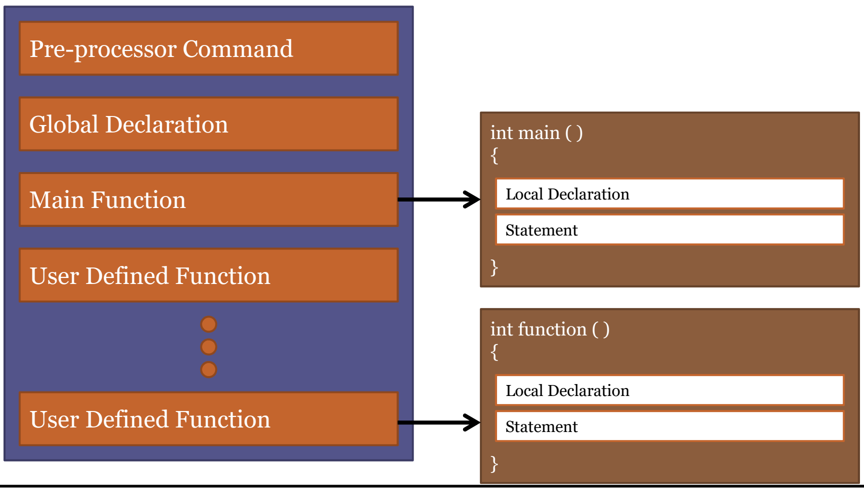
- was originally designed by Dennis Ritchie of Bell Laboratories in 1972
- Was first used as the systems language for the UNIX operating system.
- Overcomes the limitations of B



Why C?

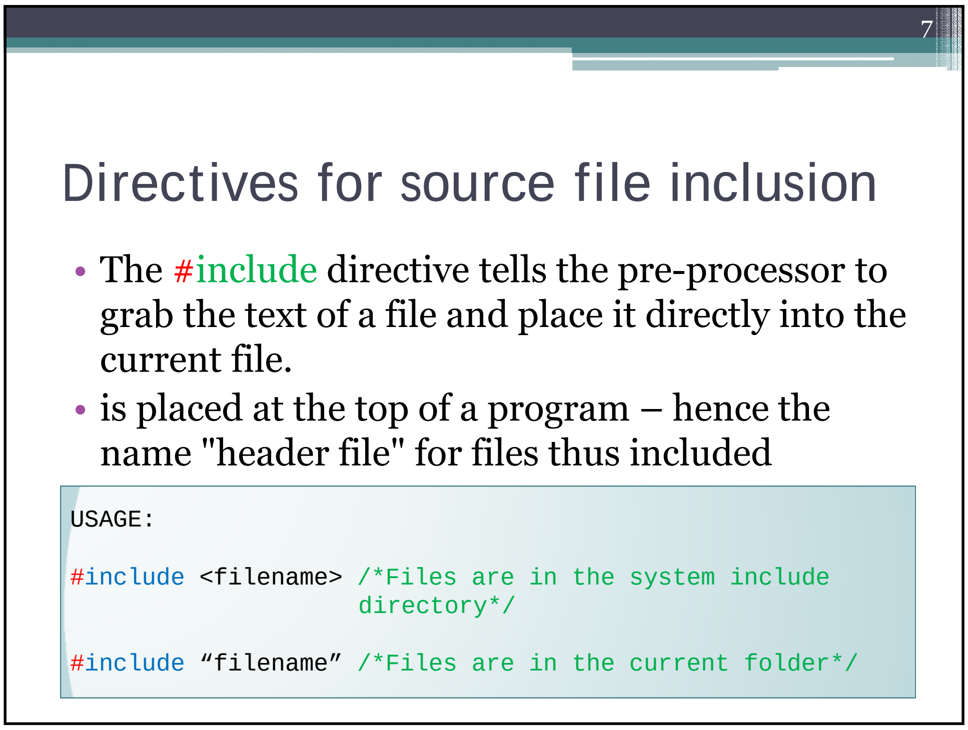
- Small – fewer keywords
- Native language of UNIX
    - UNIX is a major interactive operating system on workstations, servers, and mainframes
- The standard development language for PC
- Terse – powerful set of operators and allows us to access the machine at the bit level
- Basis of C++ and Java

C Basic Structure



Pre-processor Command

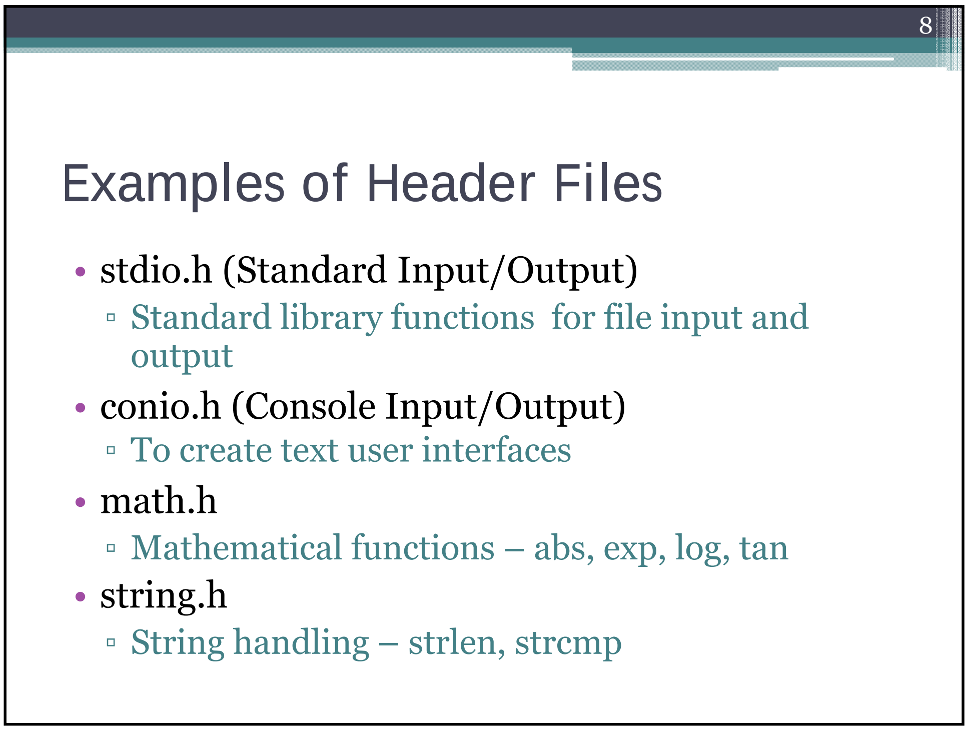
- C pre-processor modifies a source code file before handing it over to the compiler
- 3 main uses of pre-processor
  - Directives for source file inclusion (#include)
  - Macro definitions (#define)
  - Conditional inclusion (#if, …)



Directives for source file inclusion

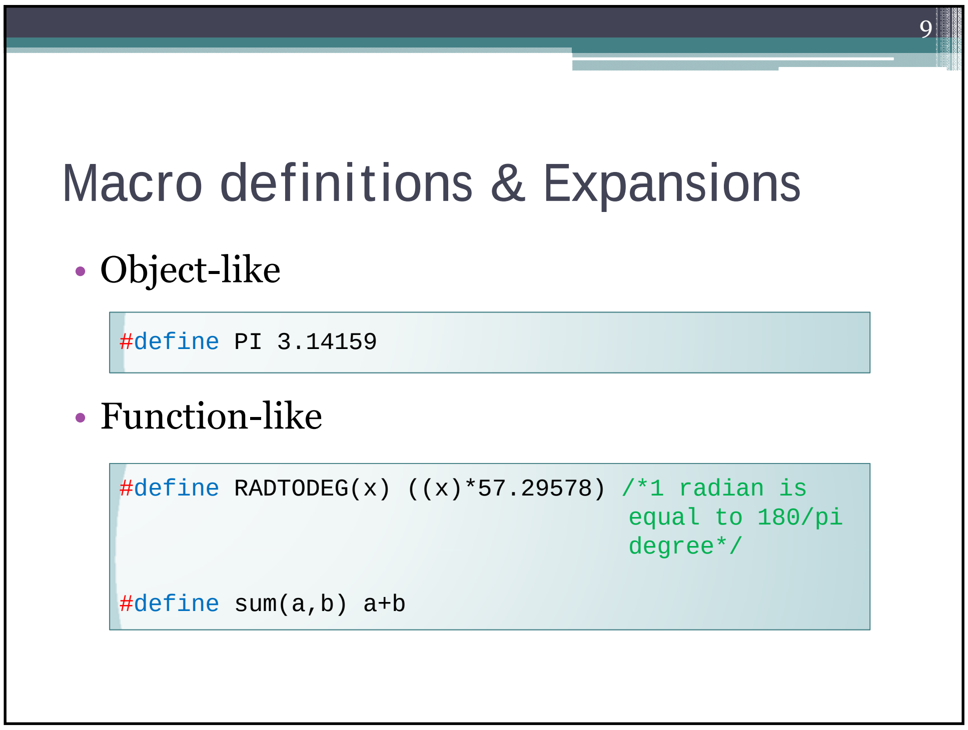
- The #include directive tells the pre-processor to grab the text of a file and place it directly into the current file.
- is placed at the top of a program – hence the name "header file" for files thus included

```
USAGE:

#include <filename> /*Files are in the system include
                    directory*/

#include “filename” /*Files are in the current folder*/
```



Examples of Header Files

- stdio.h (Standard Input/Output)
  - Standard library functions for file input and output
- conio.h (Console Input/Output)
  - To create text user interfaces
- math.h
  - Mathematical functions – abs, exp, log, tan
- string.h
  - String handling – strlen, strcmp



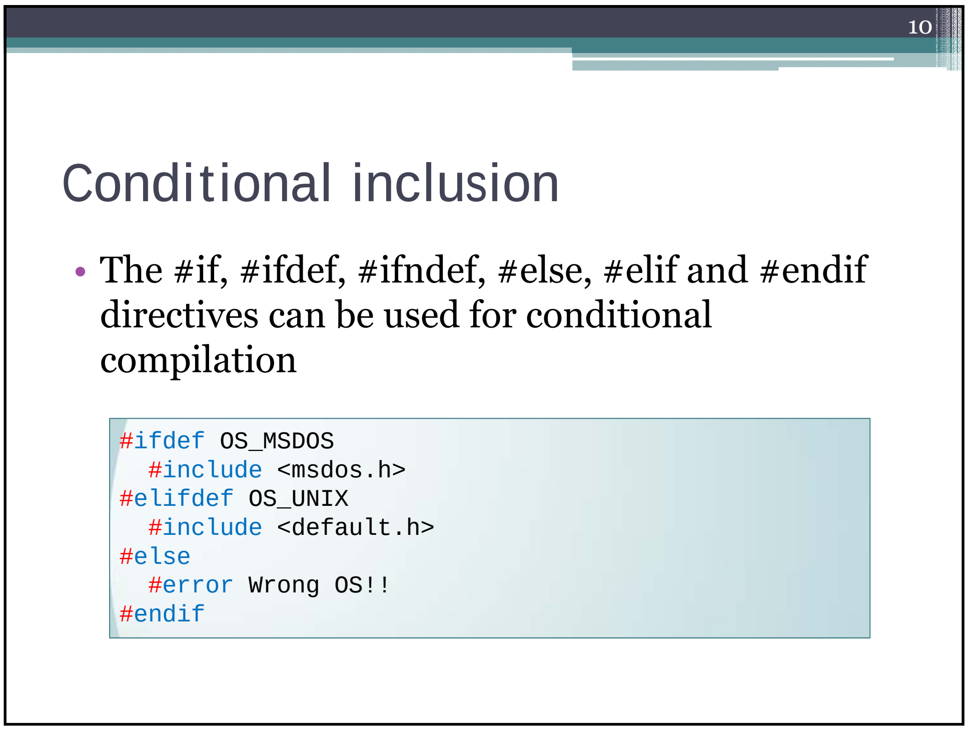
Macro definitions & Expansions

- Object-like

```
#define PI 3.14159
```

- Function-like

```
#define RADTODEG(x) ((x)*57.29578) /*1 radian is
                                    equal to 180/pi
                                    degree*/

#define sum(a,b) a+b
```

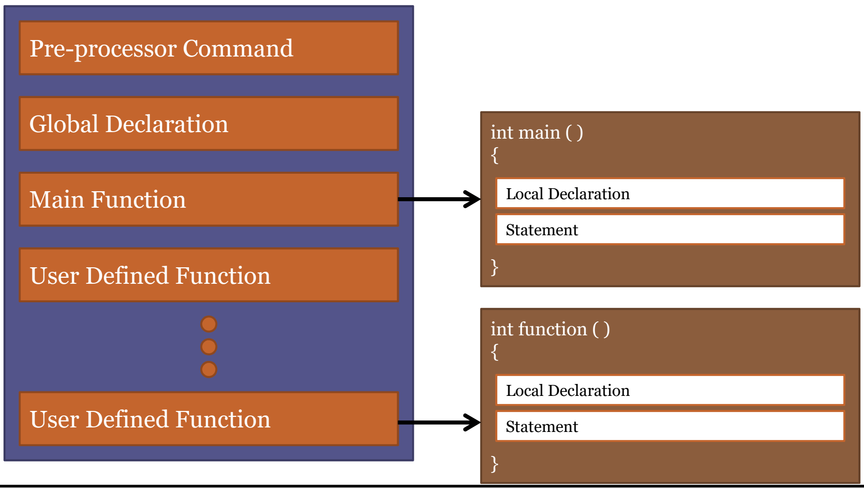


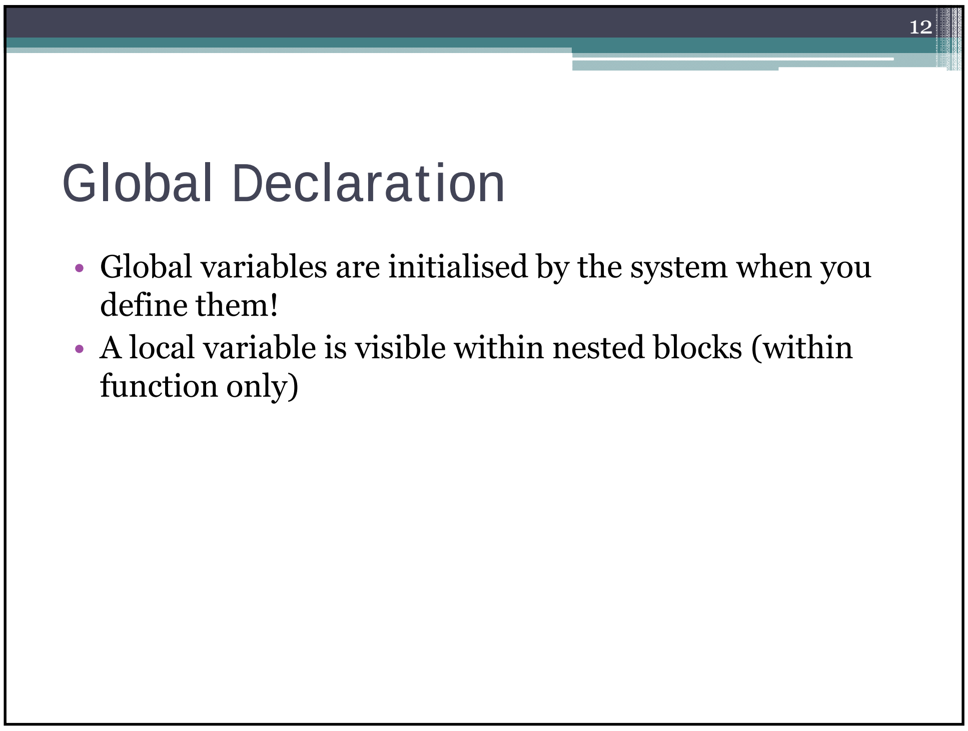
Conditional inclusion

- The #if, #ifdef, #ifndef, #else, #elif and #endif directives can be used for conditional compilation

```
#ifdef OS_MSDOS
  #include <msdos.h>
#elifdef OS_UNIX
  #include <default.h>
#else
  #error Wrong OS!!
#endif
```

C Basic Structure





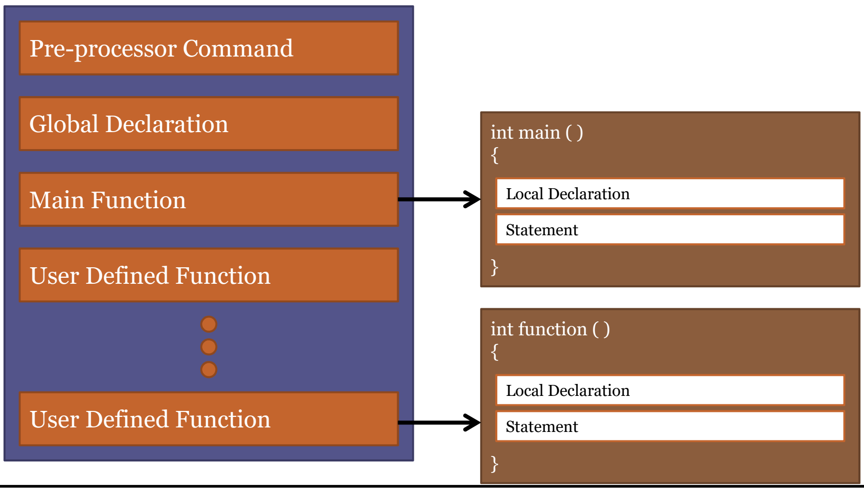
Global Declaration

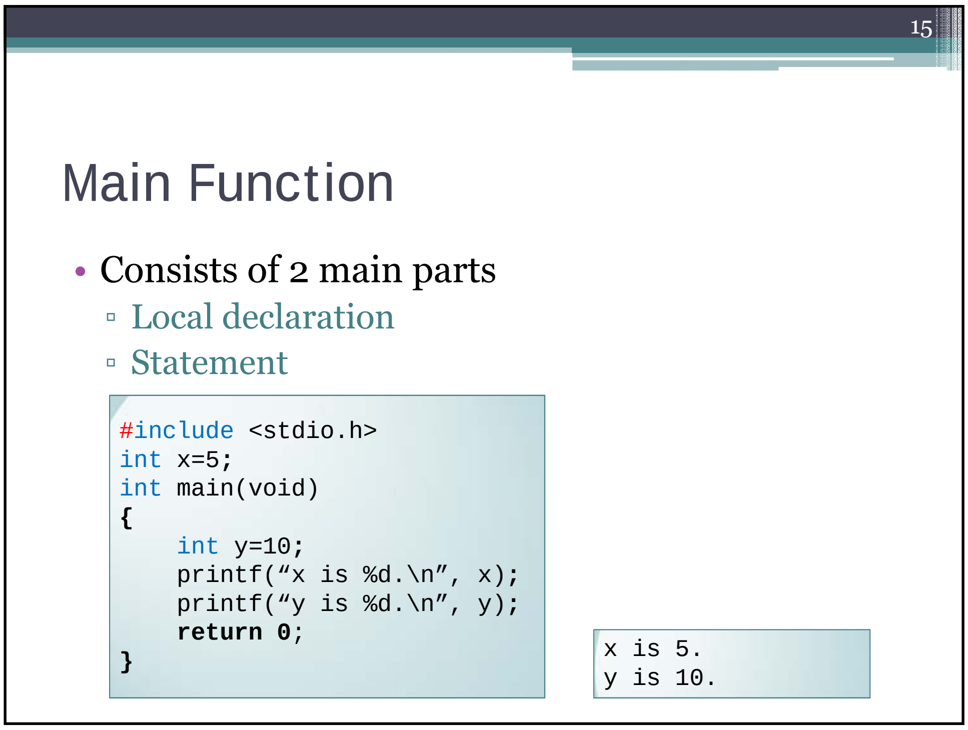
- Global variables are initialised by the system when you define them!
- A local variable is visible within nested blocks (within function only)

Example

```
#include <stdio.h>
int i=4;                        // Global definition
void main()
{
   i++;                         // Incremental
   printf("i is %d. \n", i);   // Display value of i (i=5)
   funct();
}
int funct()
{
   int i=10;                    // local definition
   i++;                         // Incremental
   printf("i is %d. \n", i);   // Display value of i (i=11)
   return 0;
}
```

C Basic Structure





15

Main Function

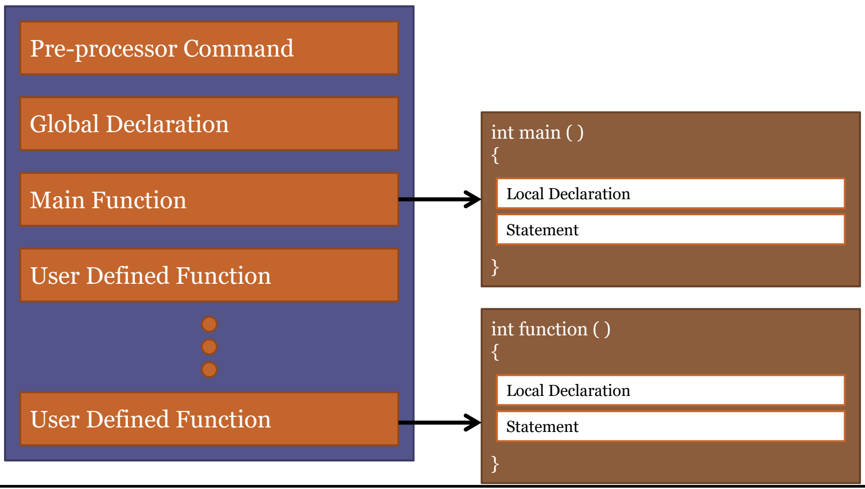
- Consists of 2 main parts
  - Local declaration
  - Statement

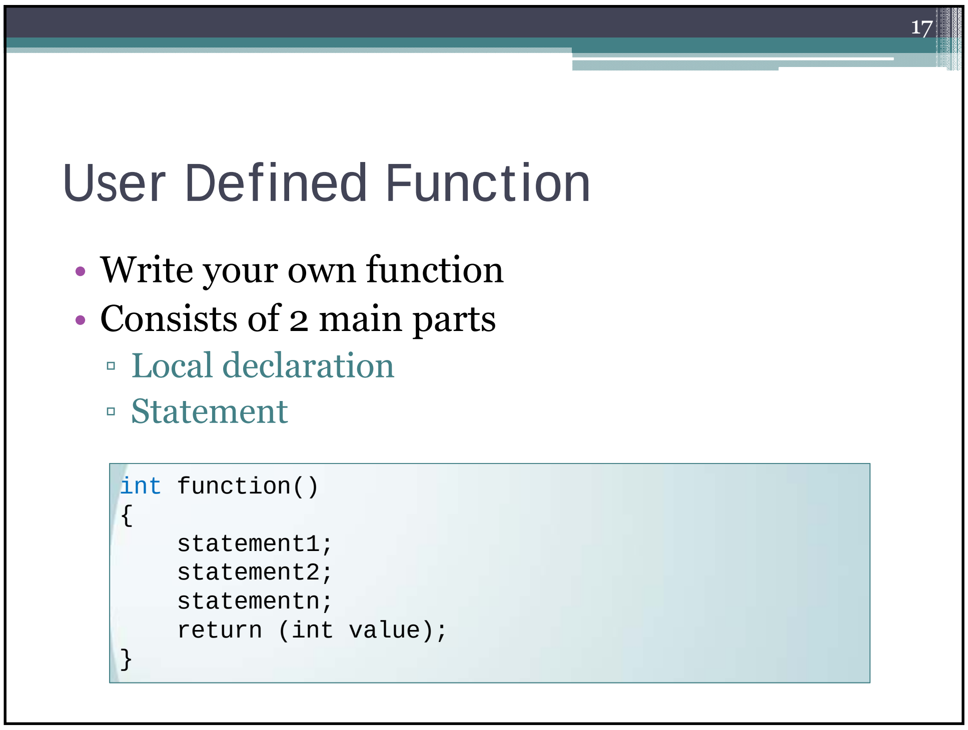
```
#include <stdio.h>
int x=5;
int main(void)
{
    int y=10;
    printf("x is %d.\n", x);
    printf("y is %d.\n", y);
    return 0;
}
```

```
x is 5.
y is 10.
```

16

C Basic Structure

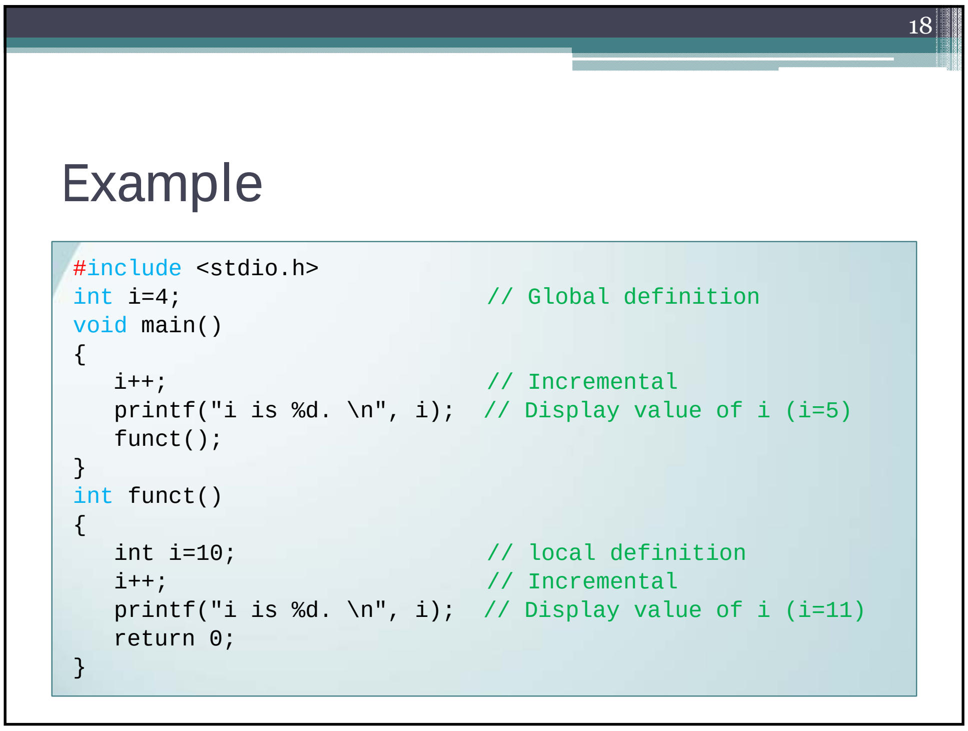




User Defined Function

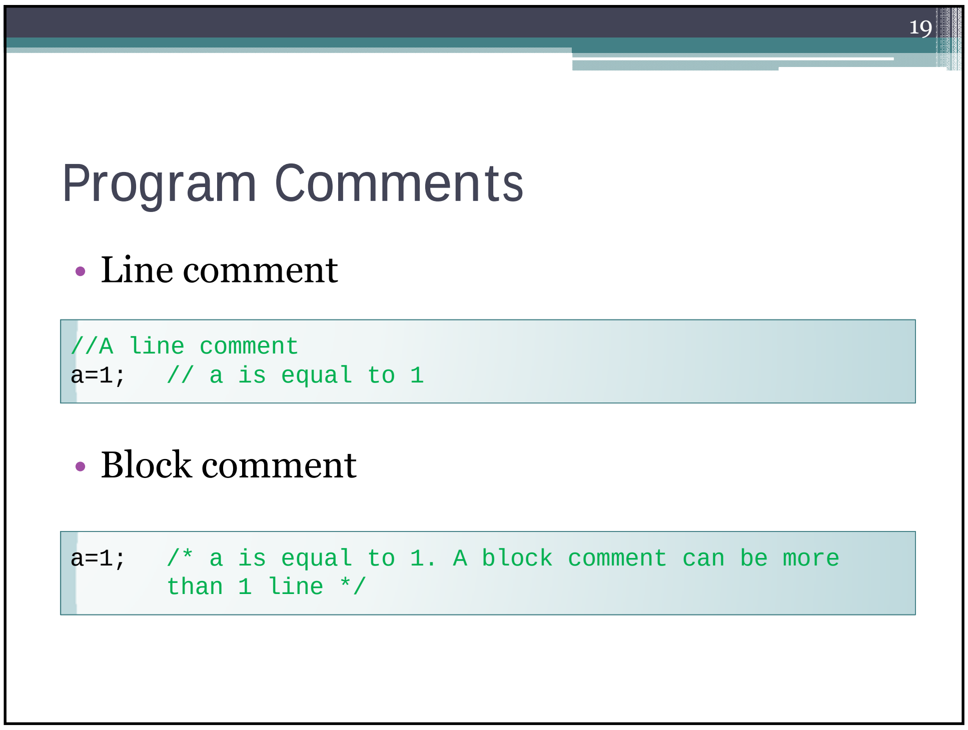
- Write your own function
- Consists of 2 main parts
  - Local declaration
  - Statement

```
int function()
{
    statement1;
    statement2;
    statementn;
    return (int value);
}
```



Example

```
#include <stdio.h>
int i=4;                        // Global definition
void main()
{
   i++;                         // Incremental
   printf("i is %d. \n", i);   // Display value of i (i=5)
   funct();
}
int funct()
{
   int i=10;                    // local definition
   i++;                         // Incremental
   printf("i is %d. \n", i);   // Display value of i (i=11)
   return 0;
}
```



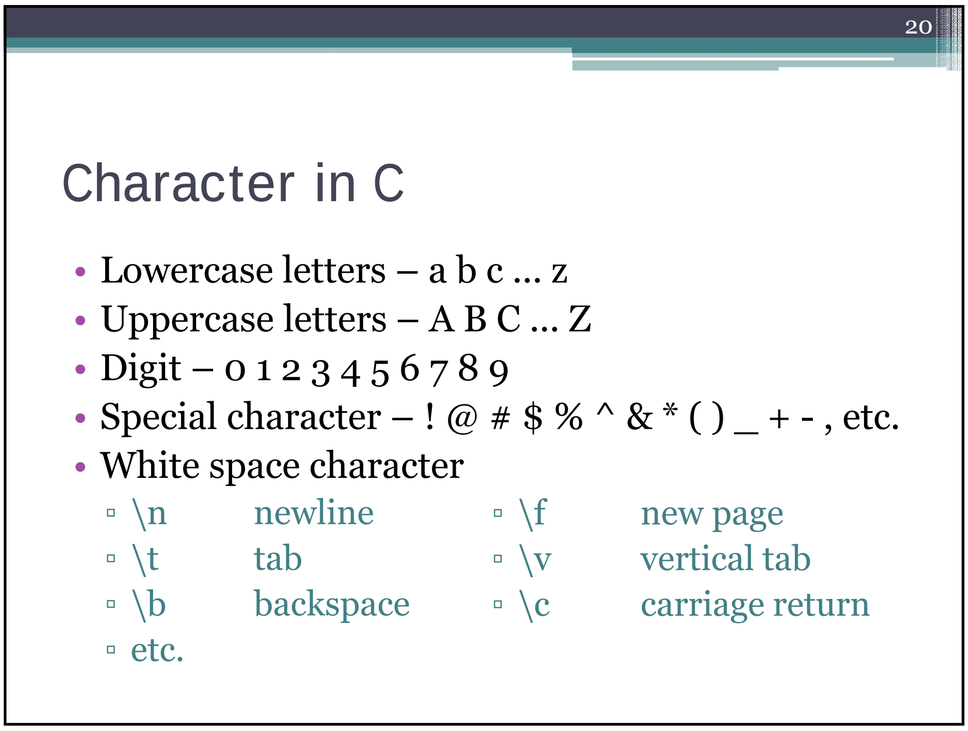
Program Comments

- Line comment

```
//A line comment
a=1;   // a is equal to 1
```

- Block comment

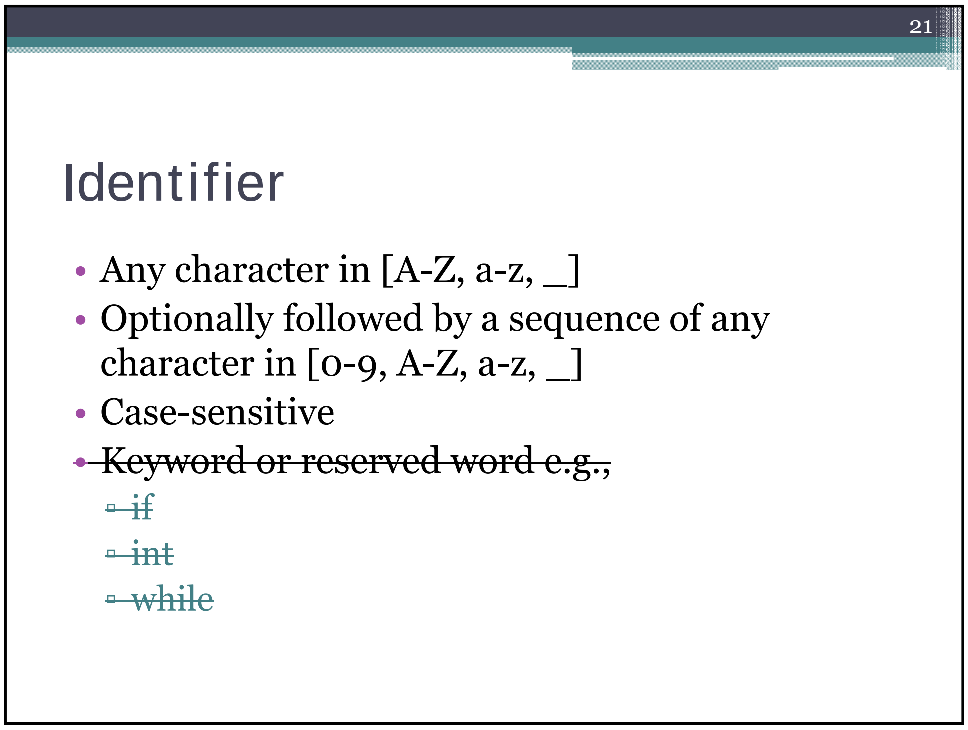
```
a=1;   /* a is equal to 1. A block comment can be more
       than 1 line */
```



Character in C

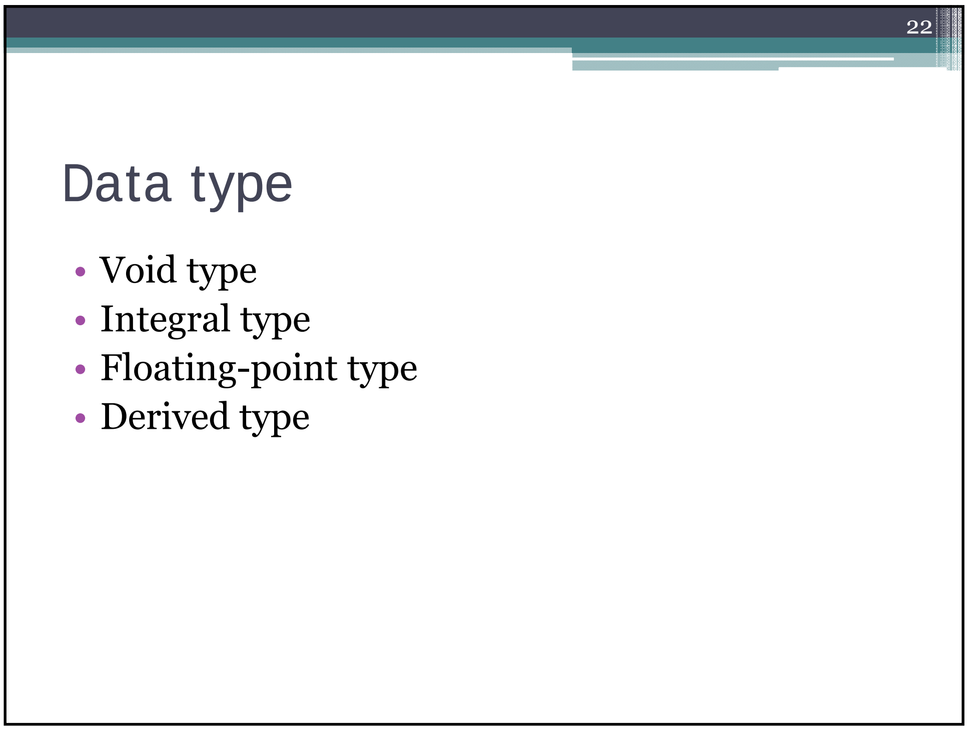
- Lowercase letters – a b c … z
- Uppercase letters – A B C … Z
- Digit – 0 1 2 3 4 5 6 7 8 9
- Special character – ! @ # $ % ^ & * ( ) _ + - , etc.
- White space character

\n	newline	\f	new page
\t	tab	\v	vertical tab
\b	backspace	\c	carriage return
etc.			



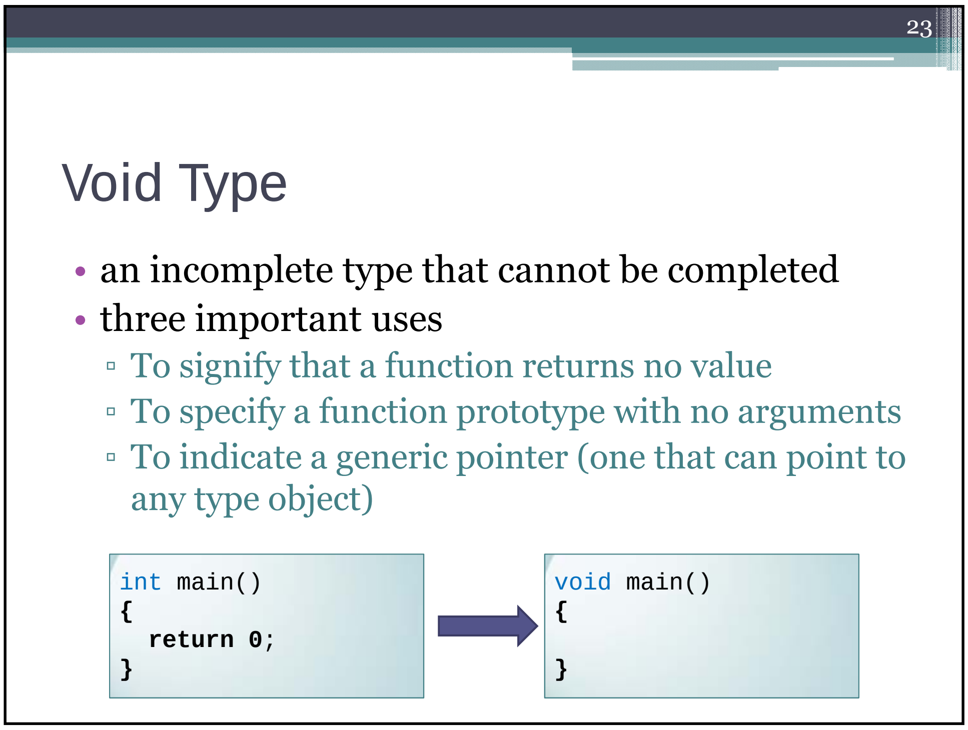
Identifier

- Any character in [A-Z, a-z, _]
- Optionally followed by a sequence of any character in [0-9, A-Z, a-z, _]
- Case-sensitive
- ~~Keyword or reserved word e.g.,~~
  - ~~if~~
  - ~~int~~
  - ~~while~~



Data type

- Void type
- Integral type
- Floating-point type
- Derived type



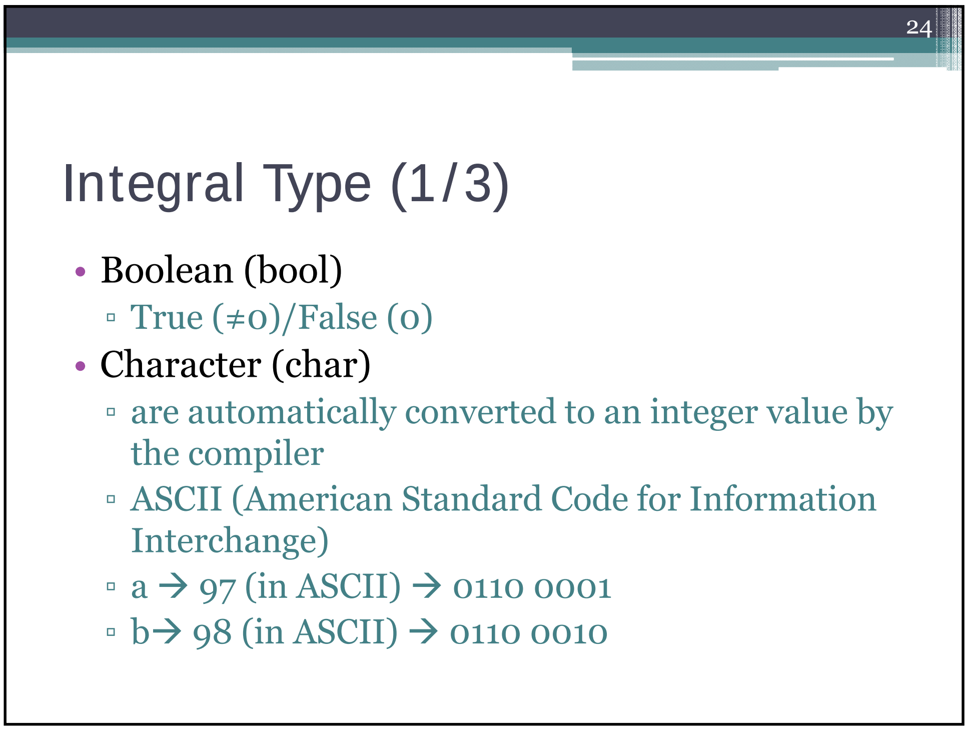
Void Type

- an incomplete type that cannot be completed
- three important uses
  - To signify that a function returns no value
  - To specify a function prototype with no arguments
  - To indicate a generic pointer (one that can point to any type object)

```
int main()
{
  return 0;
}
```

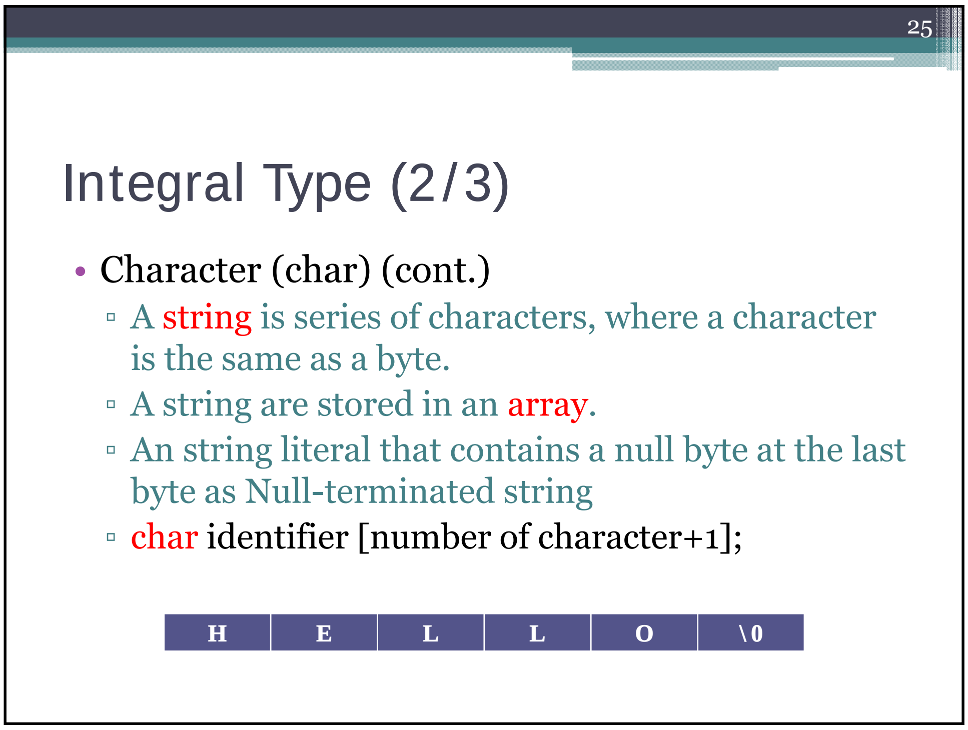
$\Rightarrow$

```
void main()
{

}
```



Integral Type (1/3)

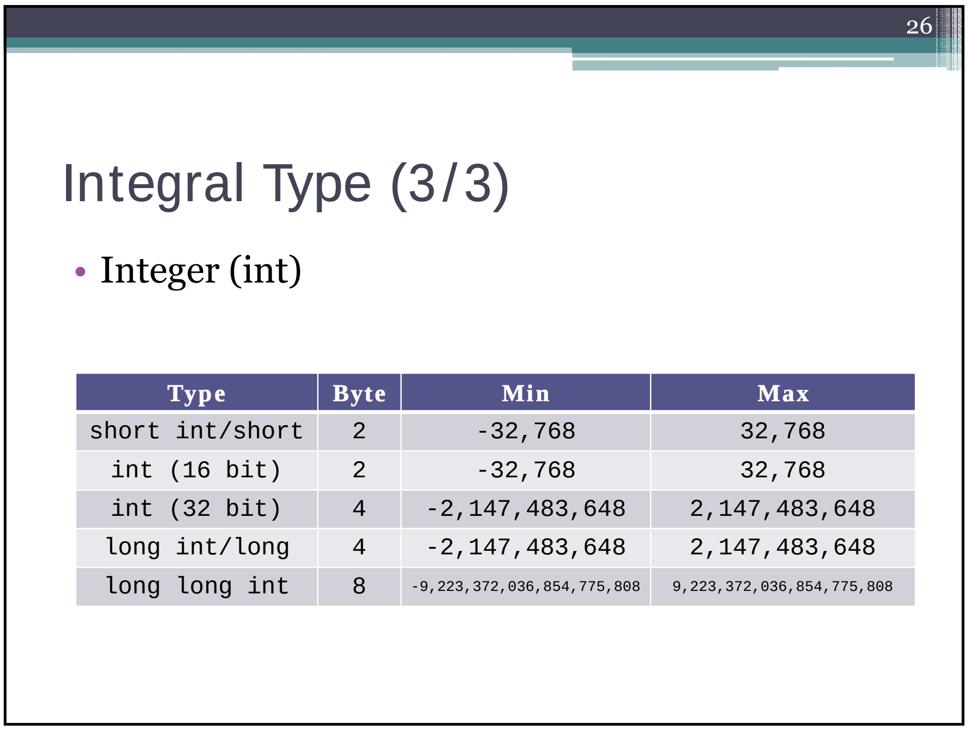
- Boolean (bool)
  - True (≠0)/False (0)
- Character (char)
  - are automatically converted to an integer value by the compiler
  - ASCII (American Standard Code for Information Interchange)
  - a → 97 (in ASCII) → 0110 0001
  - b→ 98 (in ASCII) → 0110 0010



Integral Type (2/3)

- Character (char) (cont.)
  - A string is series of characters, where a character is the same as a byte.
  - A string are stored in an array.
  - An string literal that contains a null byte at the last byte as Null-terminated string
  - char identifier [number of character+1];

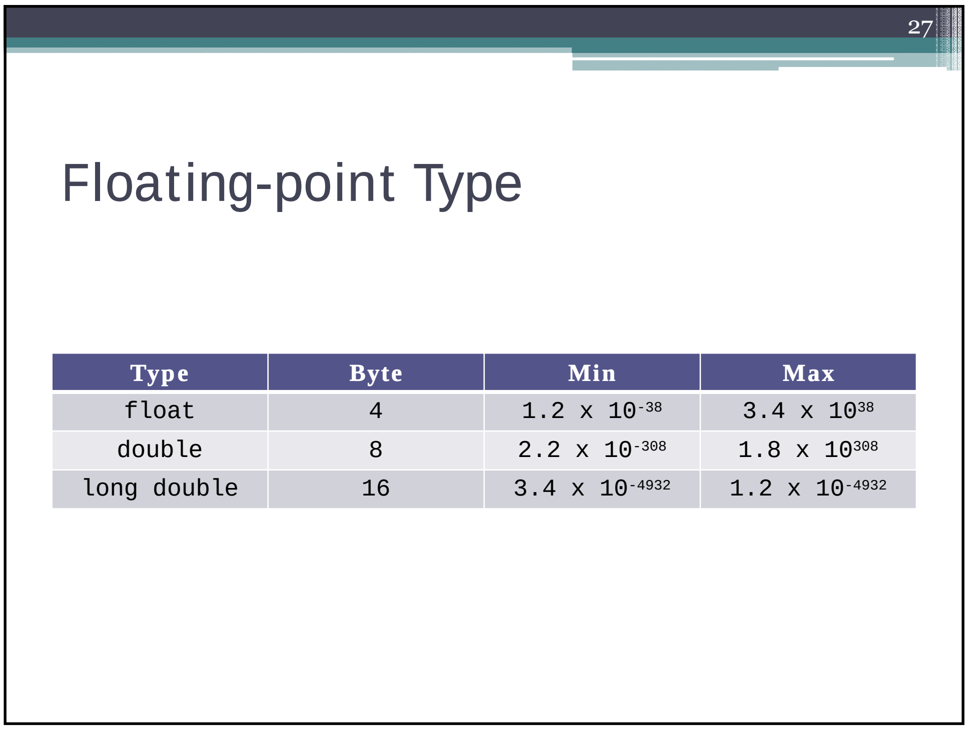
H	E	L	L	O	\0



Integral Type (3/3)

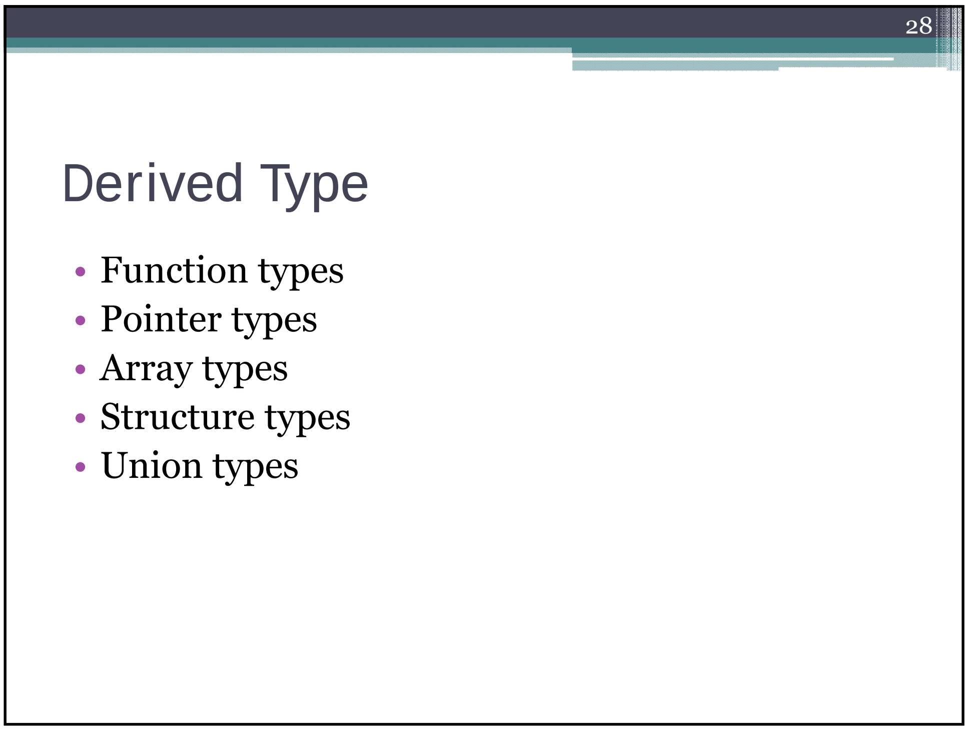
- Integer (int)

Type	Byte	Min	Max
short int/short	2	-32,768	32,768
int (16 bit)	2	-32,768	32,768
int (32 bit)	4	-2,147,483,648	2,147,483,648
long int/long	4	-2,147,483,648	2,147,483,648
long long int	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,808



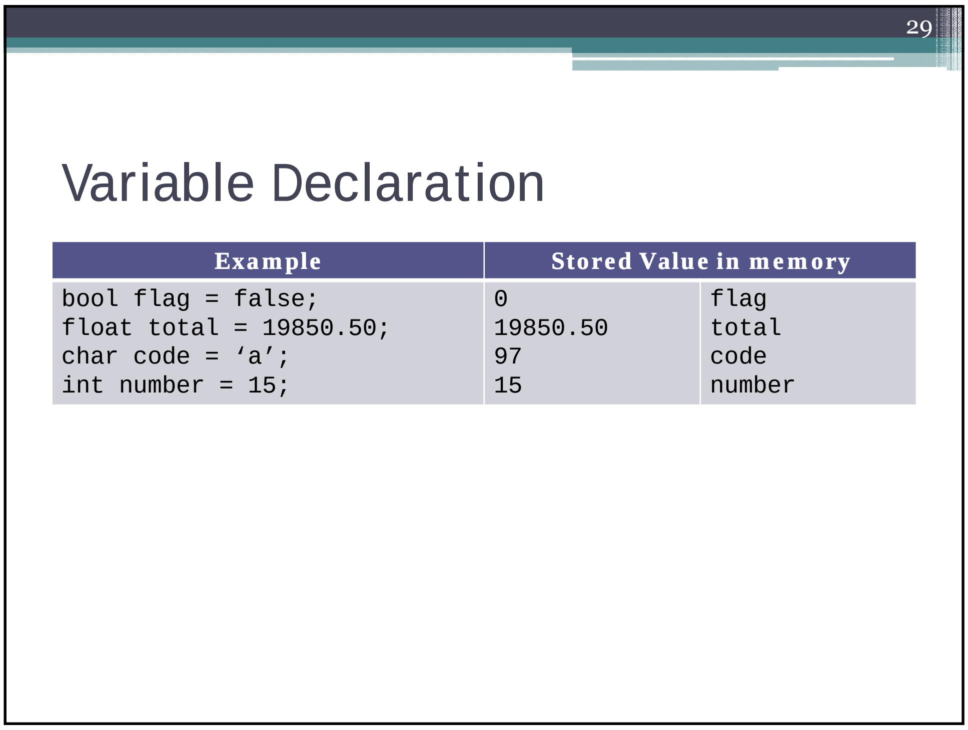
Floating-point Type

Type	Byte	Min	Max
float	4	1.2×10^{-38}	3.4×10^{38}
double	8	2.2×10^{-308}	1.8×10^{308}
long double	16	3.4×10^{-4932}	1.2×10^{-4932}



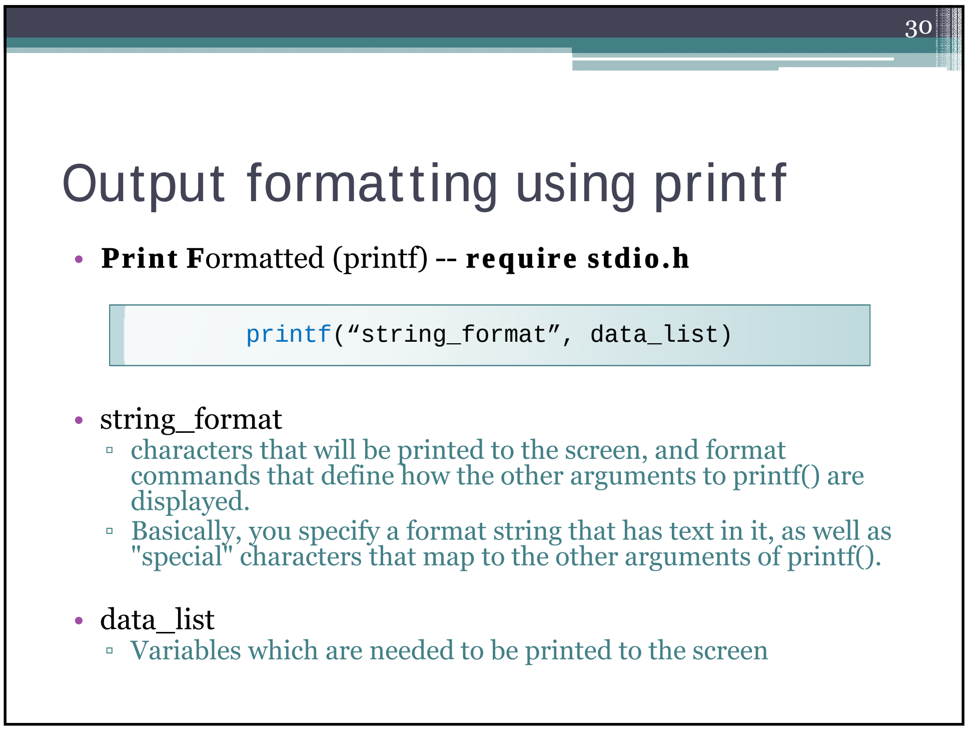
Derived Type

- Function types
- Pointer types
- Array types
- Structure types
- Union types



Variable Declaration

Example	Stored Value in memory	
`bool flag = false;`	0	flag
`float total = 19850.50;`	19850.50	total
`char code = ‘a’;`	97	code
`int number = 15;`	15	number

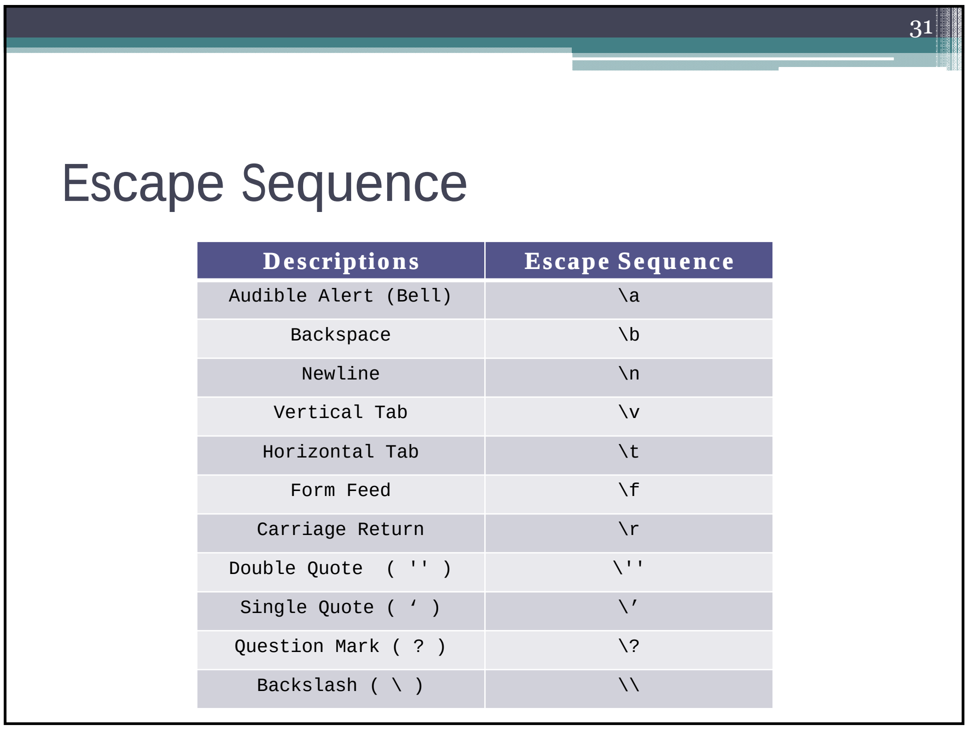


Output formatting using printf

- **Print F**ormatted (printf) -- **require stdio.h**

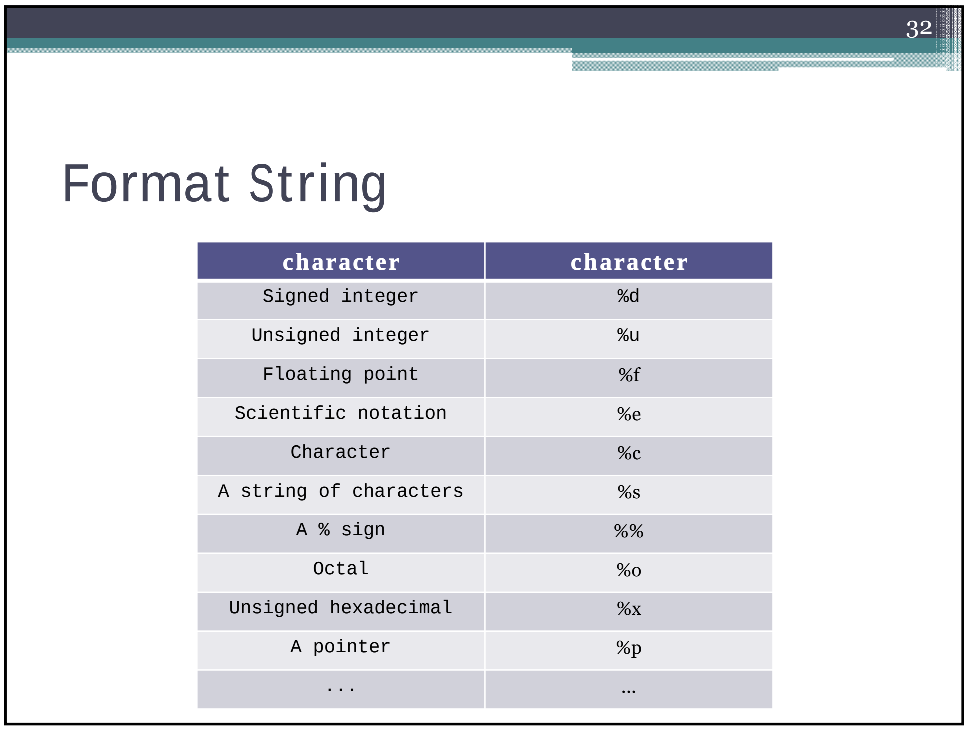
```
printf(“string_format”, data_list)
```

- string_format
  - characters that will be printed to the screen, and format commands that define how the other arguments to printf() are displayed.
  - Basically, you specify a format string that has text in it, as well as "special" characters that map to the other arguments of printf().
- data_list
  - Variables which are needed to be printed to the screen



Escape Sequence

Descriptions	Escape Sequence
Audible Alert (Bell)	\a
Backspace	\b
Newline	\n
Vertical Tab	\v
Horizontal Tab	\t
Form Feed	\f
Carriage Return	\r
Double Quote ('')	\''
Single Quote (‘)	\’
Question Mark (?)	\?
Backslash (\ )	\\



Format String

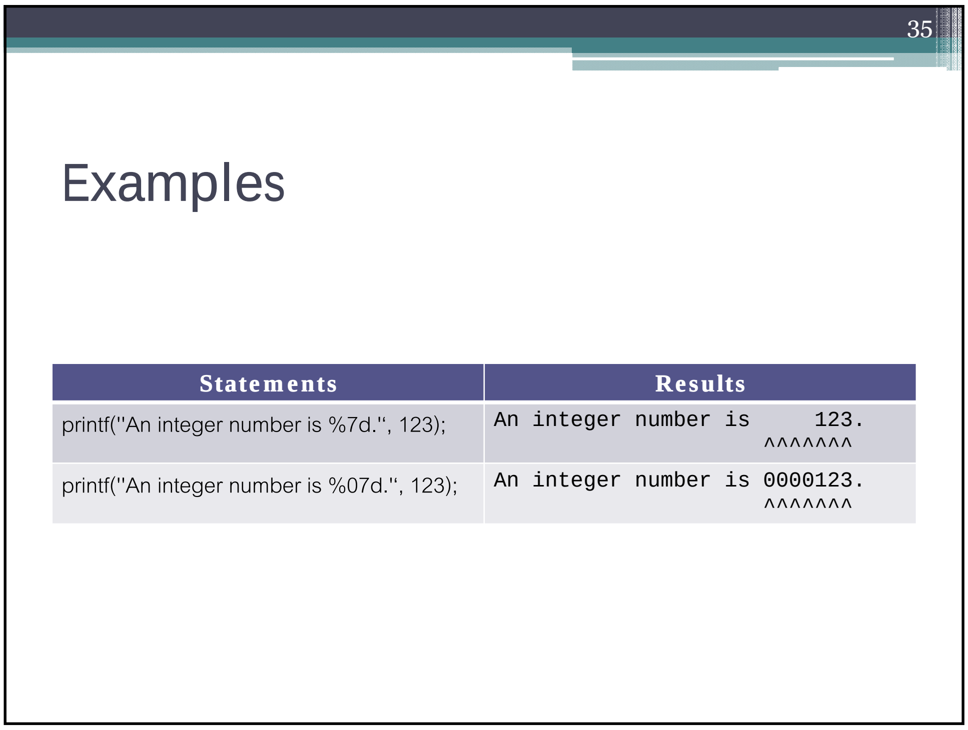
character	character
Signed integer	%d
Unsigned integer	%u
Floating point	%f
Scientific notation	%e
Character	%c
A string of characters	%s
A % sign	%%
Octal	%o
Unsigned hexadecimal	%x
A pointer	%p
...	…

Examples

Statements	Results
printf("IT.KMITL");	IT.KMITL
printf(" \"Hello\" \nHow are you\?");	"Hello" How are you?
printf("2 x 4 = %d", 8);	2 x 4 = 8
printf("Number = %f", 2.5);	Number = 2.5
printf("%d + %d = %d", a, b, a+b);	Assume that a=1, b=2. Hence, 1 + 2 = 3

Output formatting using printf

- An integer placed between a % sign and the format command acts as a minimum field width specifier, and pads the output with spaces or zeros to make it long enough.
- If you want to pad with zeros, place a zero before the minimum field width specifier. You can use a precision modifier, which has different meanings depending on the format code being used.

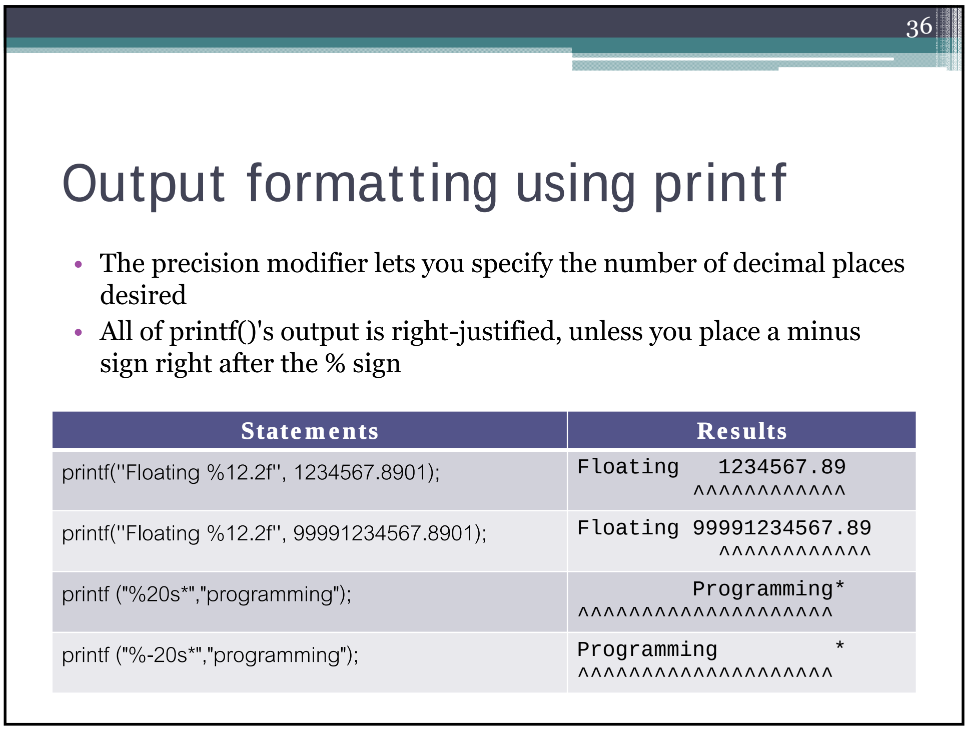


Examples

Statements	Results

printf("An integer number is %7d.", 123);	`An integer number is     123.`
	`                     ^^^^^^^`

printf("An integer number is %07d.", 123);	`An integer number is 0000123.`
	`                     ^^^^^^^`



Output formatting using printf

- The precision modifier lets you specify the number of decimal places desired
- All of printf()'s output is right-justified, unless you place a minus sign right after the % sign

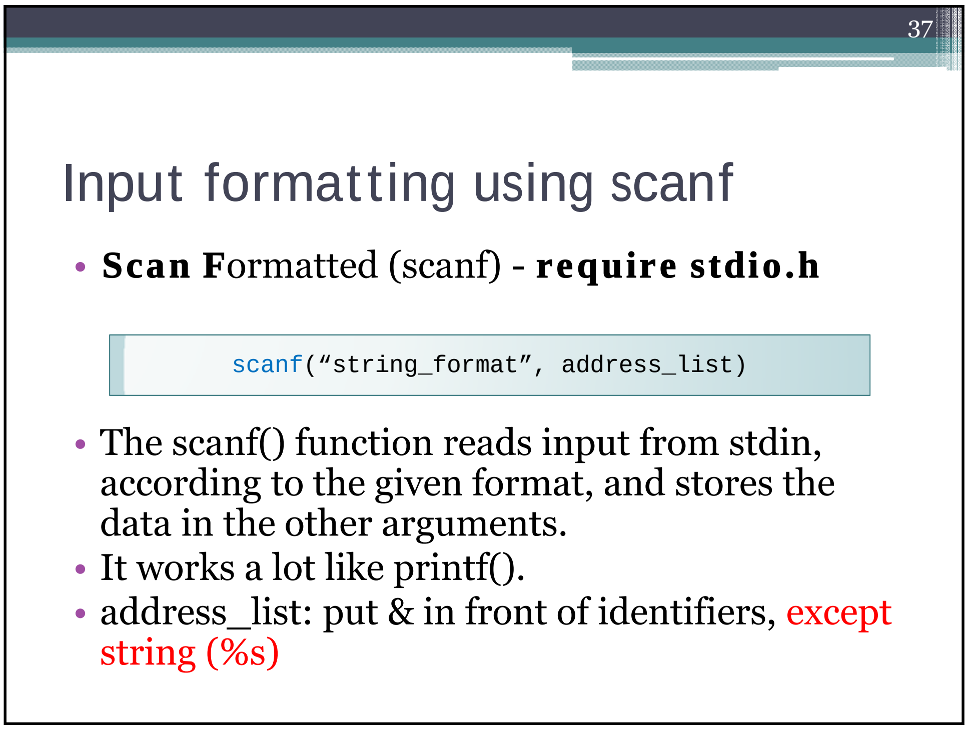
Statements	Results

printf("Floating %12.2f", 1234567.8901);	`Floating   1234567.89`
	`         ^^^^^^^^^^^^`

printf("Floating %12.2f", 99991234567.8901);	`Floating 99991234567.89`
	`           ^^^^^^^^^^^^`

printf ("%20s*","programming");	`         Programming*`
	`^^^^^^^^^^^^^^^^^^^^`

printf ("%-20s*","programming");	`Programming         *`
	`^^^^^^^^^^^^^^^^^^^^`

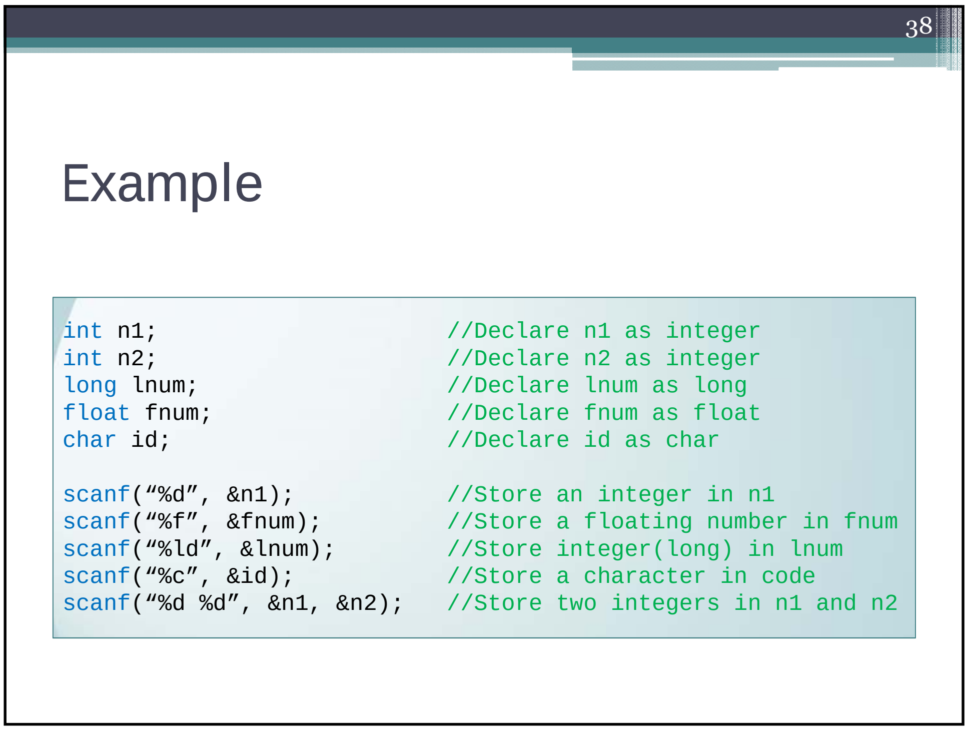


Input formatting using scanf

- **Scan F**ormatted (scanf) - **require stdio.h**

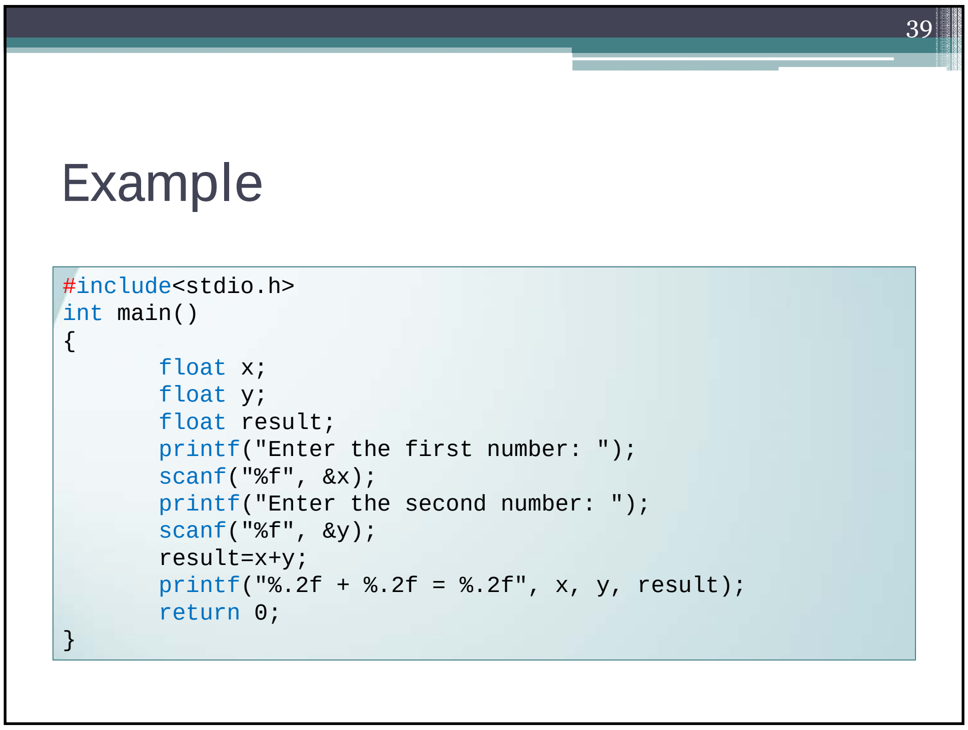
```
scanf("string_format", address_list)
```

- The scanf() function reads input from stdin, according to the given format, and stores the data in the other arguments.
- It works a lot like printf().
- address_list: put & in front of identifiers, except string (%s)



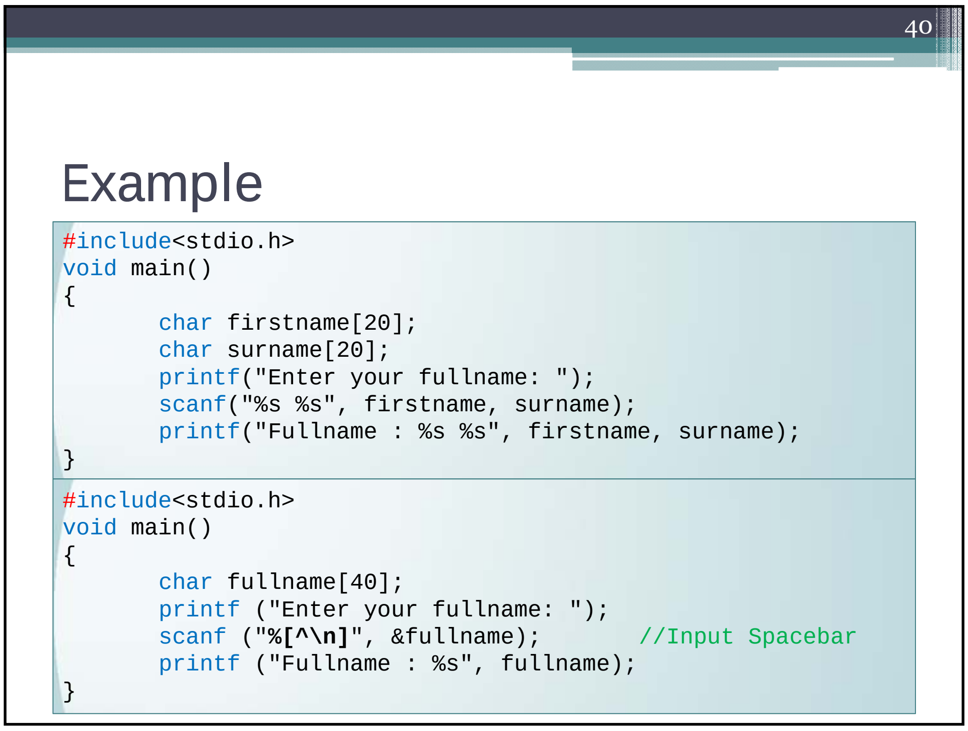
Example

```
int n1;                       //Declare n1 as integer
int n2;                       //Declare n2 as integer
long lnum;                    //Declare lnum as long
float fnum;                   //Declare fnum as float
char id;                      //Declare id as char

scanf("%d", &n1);             //Store an integer in n1
scanf("%f", &fnum);           //Store a floating number in fnum
scanf("%ld", &lnum);          //Store integer(long) in lnum
scanf("%c", &id);             //Store a character in code
scanf("%d %d", &n1, &n2);     //Store two integers in n1 and n2
```



Example

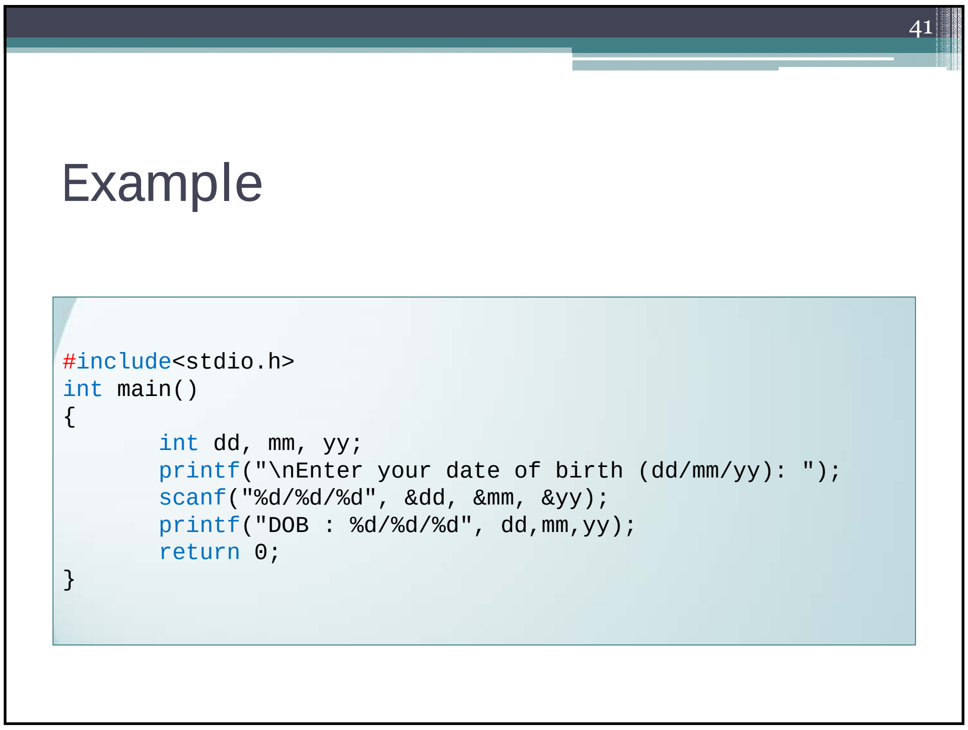
```c
#include<stdio.h>
int main()
{
	float x;
	float y;
	float result;
	printf("Enter the first number: ");
	scanf("%f", &x);
	printf("Enter the second number: ");
	scanf("%f", &y);
	result=x+y;
	printf("%.2f + %.2f = %.2f", x, y, result);
	return 0;
}
```



Example

```c
#include<stdio.h>
void main()
{
	char firstname[20];
	char surname[20];
	printf("Enter your fullname: ");
	scanf("%s %s", firstname, surname);
	printf("Fullname : %s %s", firstname, surname);
}
```

```c
#include<stdio.h>
void main()
{
	char fullname[40];
	printf ("Enter your fullname: ");
	scanf ("%[^\n]", &fullname);       //Input Spacebar
	printf ("Fullname : %s", fullname);
}
```



Example

```c
#include<stdio.h>
int main()
{
	int dd, mm, yy;
	printf("\nEnter your date of birth (dd/mm/yy): ");
	scanf("%d/%d/%d", &dd, &mm, &yy);
	printf("DOB : %d/%d/%d", dd,mm,yy);
	return 0;
}
```