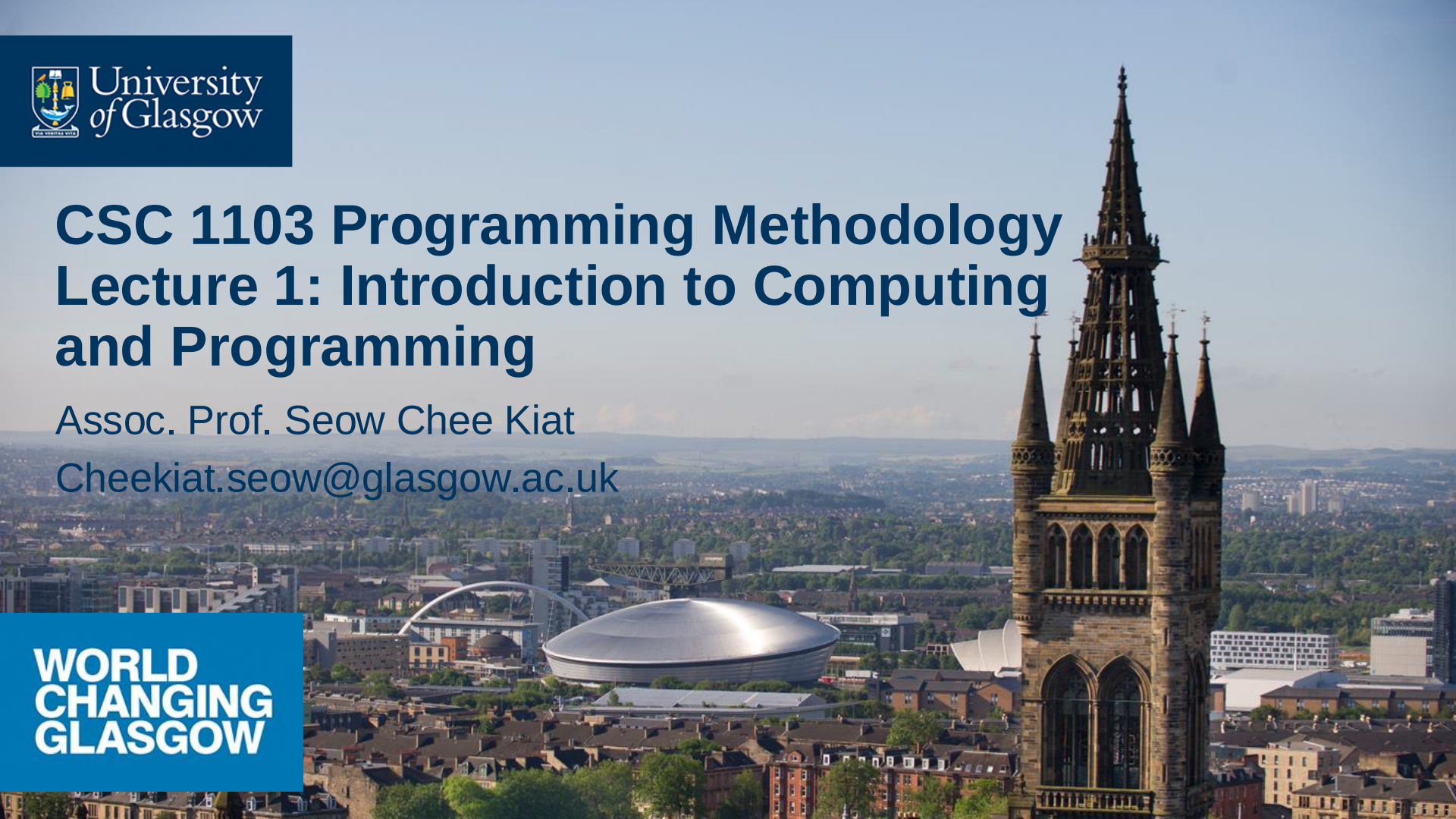


CSC 1103 Programming Methodology Lecture 1: Introduction to Computing and Programming

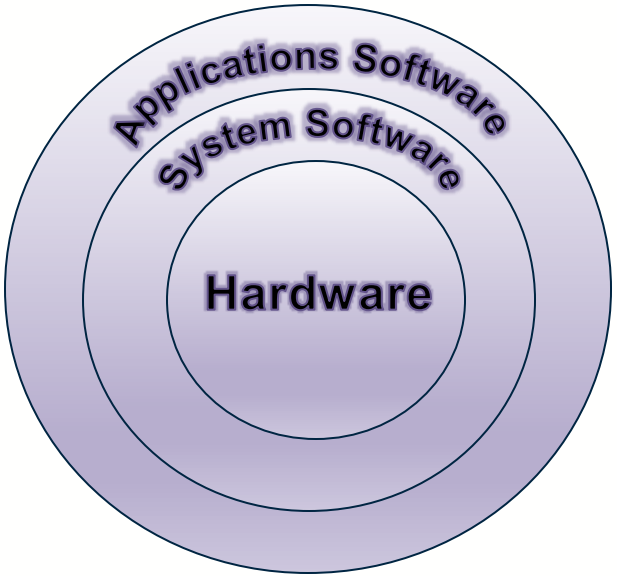
Assoc. Prof. Seow Chee Kiat

Cheekiat.seow@glasgow.ac.uk

University of Glasgow School of Computing Science

Architecture of Computing System

- Applications Software (AS)
  - solve a particular problem or carry out a specific task
    - Productivity Software such as Microsoft Words, Excel
    - Social Media software such as WhatsApps, WeChat, Facebook etc
- System Software (SS)
  - Coordinates the activities and function of hardware and software and it controls the operations of computer hardware.
  - Many type of SS such as operating system, compilers, utility software, device drivers and firmware
    - Two important SS
      - Operating system such as Mac OS, Windows and Linux
      - Compilers such as GCC, MinGW, Cygwin



University of Glasgow
School of Computing Science

Architecture of Computing System

- Hardware

University of Glasgow School of Computing Science

Programming Languages

- Electronic computer hardware only knows two Binary digit or bit 1 and 0. Computers listen to commands which are called instructions. Instructions are collections of bits that computers understands and obeys. For example

  1000110010100000

  tells one computer to add two numbers.

- Imagine write two instructions of adding two numbers twice

  10001100101000001000110010100000 0

  This is called Machine Language Programming.

University of Glasgow School of Computing Science

Programming Languages

- Assembly Language Programming use symbolic notation and use a program called assembler to translate symbolic version of instruction into binary version machine code. For example, the assembly language programmer would write

  add A, B

  and the assembler will translate the notation to 1000110010100000

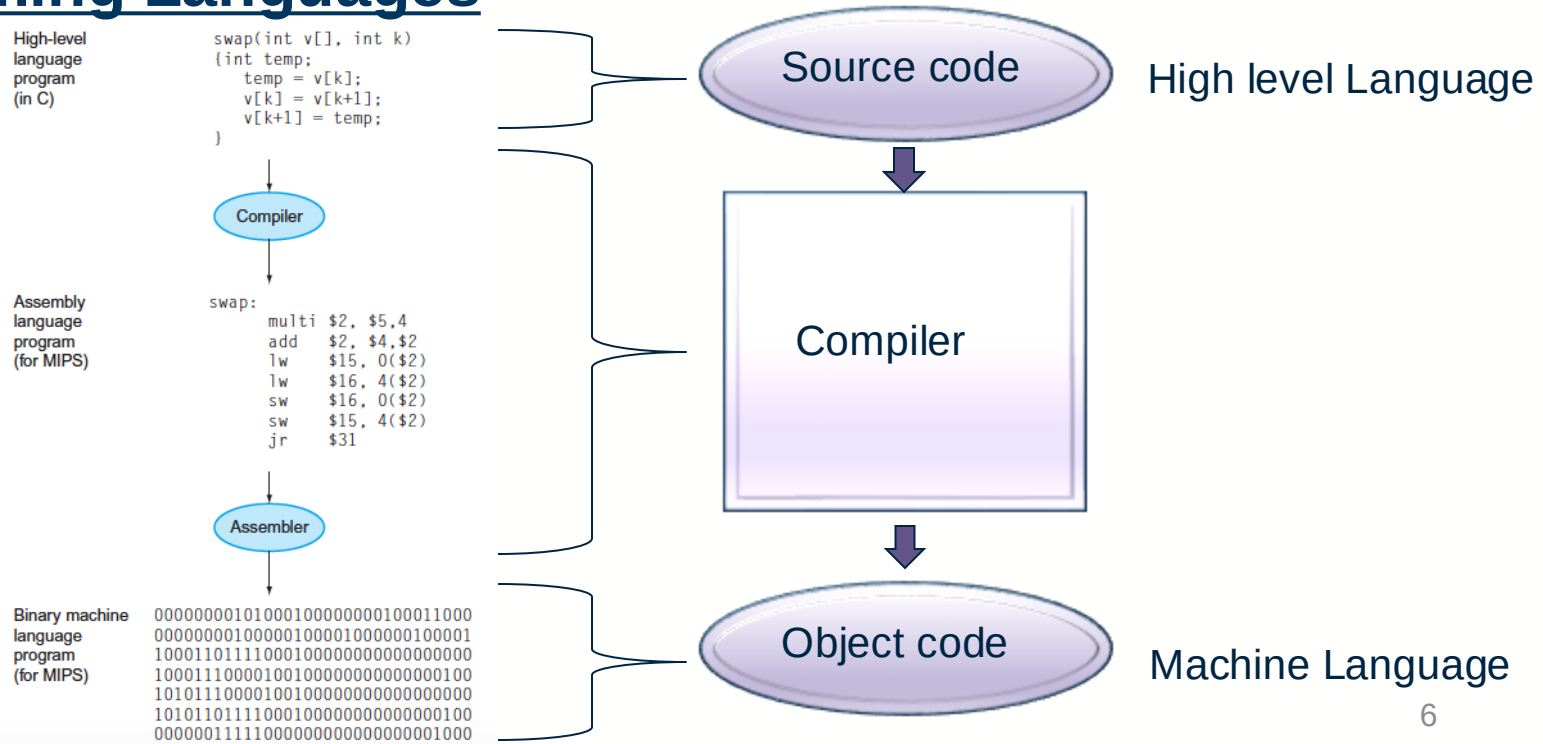
- High Level Language Programming such as C Language use even more closer to English Language that is more aligned to problem that is going to be formulated and resolved such as

  A+B instead of add A, B

  - Use a compiler to translate from the high level language to assembly language

University of Glasgow School of Computing Science

Programming Languages



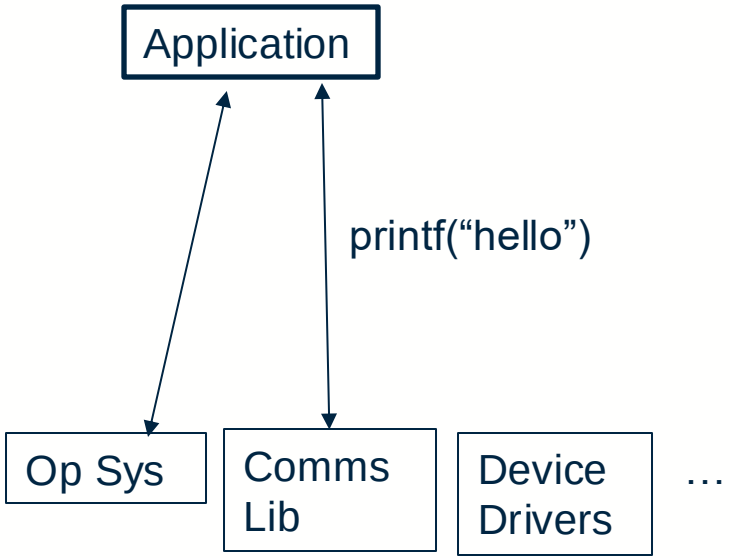
University of Glasgow School of Computing Science

Programming Language

- Application Level

Languages	Facilities
Java Python Haskell ...	Strong Types Automatic Memory Management Abstract Resources IDE

- System Level
  - Languages: C, Rust, Ocaml
  - Construct :
    - Operating Systems
    - Performance-critical libraries, e.g. Communication
      Numerical, e.g. BLAS
    - Device Drivers



Why C Language?

- C/C++ are the most heavily used programming language in industry, so the ability to use it well is essential for your career
- C (or its sister language, C++) is the language for operating systems and embedded systems– mastery of C, especially pointers and string handling, is **essential** for success in Networked System and Operating System. Hence in the Internet of Things (IoT programming.
- C syntax and libraries have strongly influenced **many** other languages like Java, JavaScript, Go, Rust, …
- All real software is built in multiple languages, so languages typically offer a foreign function interface (FFI) to call out into external libraries, e.g. Java Native Interface (JNI). Some FFI support multiple languages, but C is the common denominator supported by **all** FFI

University of Glasgow School of Computing Science

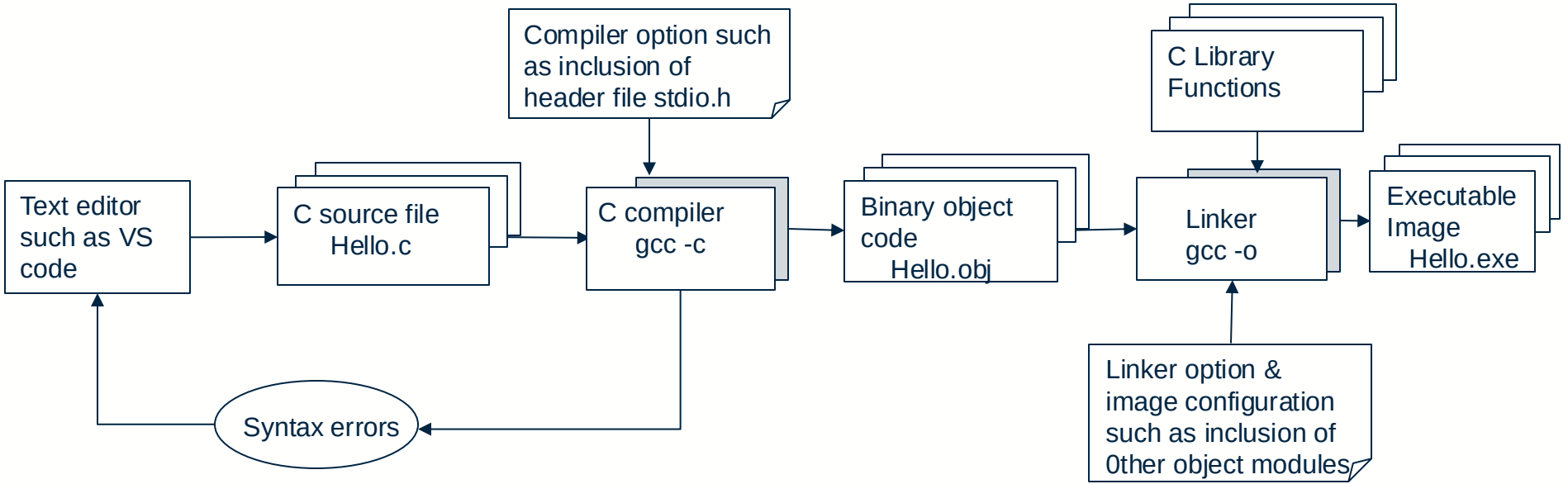
C Standard Used

- C is a standard Language
  - C89 also known as ANSI C or Standard C
  - C99 ISO standard
  - C11 ISO standard
- This course focus on
  - Primarily on C89
  - But use C11 in the lab as GNU C Compiler (GCC)
    - Standard compliance not enforced (compiler option is std GNU-11)
  - Why not C11? Not Fully implemented, no good standard book.

University of Glasgow School of Computing Science

Edit-Compile-Link-Execute (ECLC) Cycle

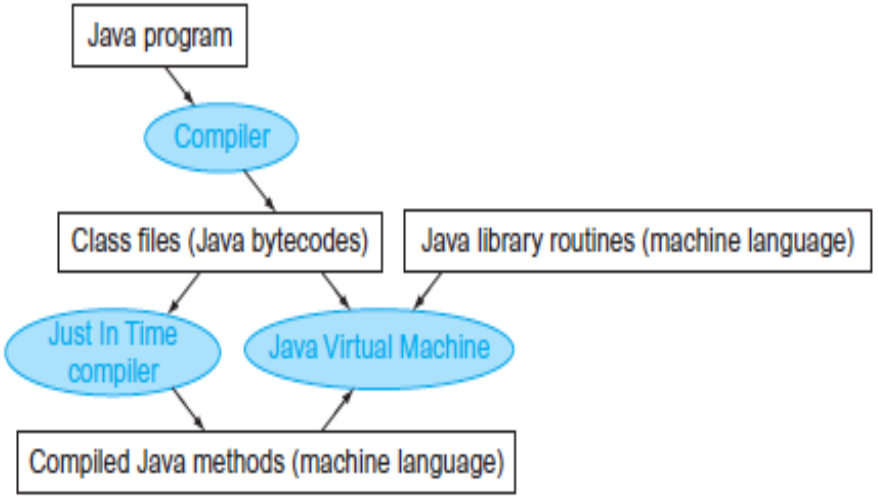
- gcc stands for GNU (not Unix) compiler collection



University of Glasgow School of Computing Science

Edit-Compile-Link-Execute (ECLC) Cycle

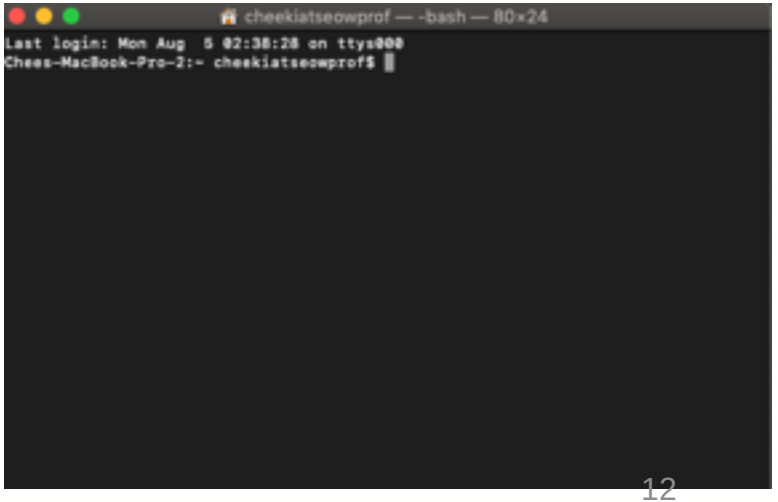
- For Java programming which use Eclipse that comes with compiler/linker
  - When you edited classes in Eclipse, behind the scene Eclipse was busy converting java program to Java bytecodes which are instructions that Java Virtual Machine (JVM) knows how to execute
  - When you requested that Eclipse run a class that contained main(), it issued appropriate commands to Windows to make this happen



University of Glasgow
School of Computing Science

Edit-Compile-Link-Execute (ECLC) Cycle

- For Linux programming using most programming languages, you have to explicitly execute commands within your chosen shell for editing, translating, and executing your programs:
  - Explicit editing commands: vi, emacs, gedit, <others>
  - Explicit compilation commands: gcc, <others>
  - Explicit linking commands: gcc, <others>
  - Explicit execution commands: ./program_name



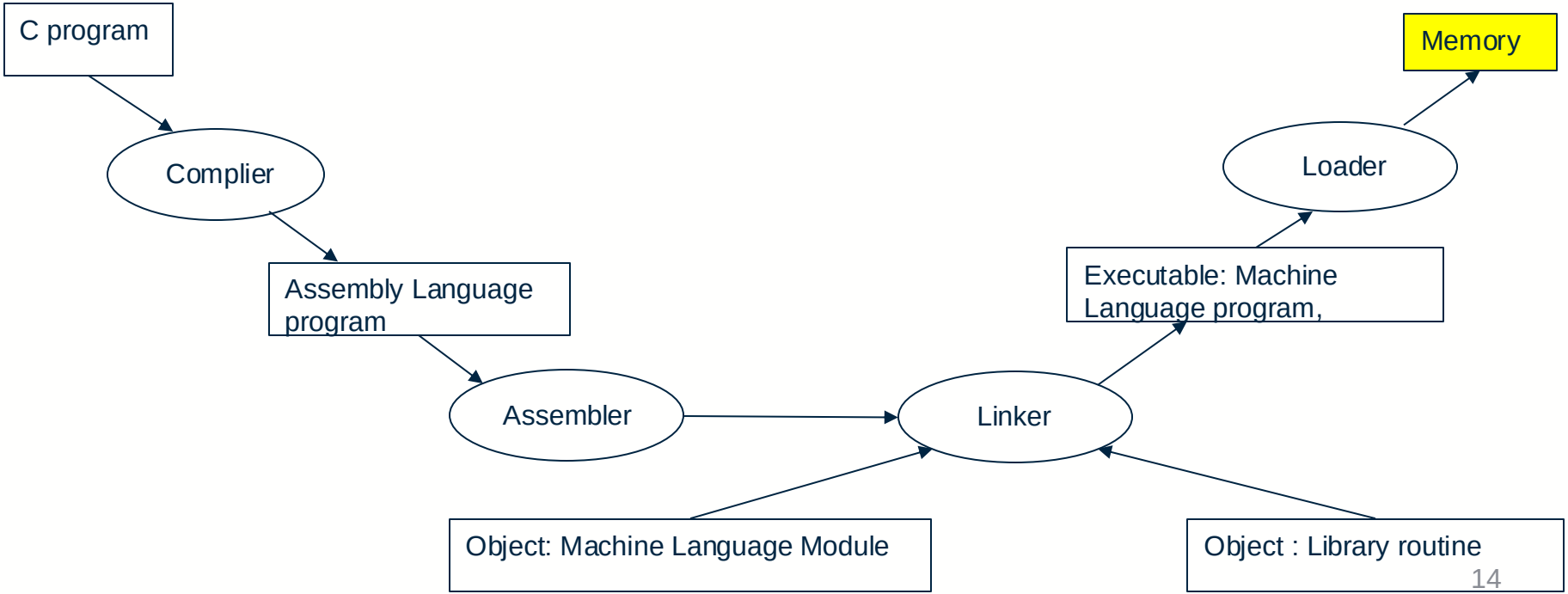
University of Glasgow School of Computing Science

Edit-Compile-Link-Execute (ECLC) Cycle

- Suppose you have created a C program in a file named “Hello.c”
  - $gcc –c Hello.c
    - cause the C compiler to be invoked. If the compilation is successful, it will create a filename “Hello.o”
  - $gcc – o Hello Hello.o
    - cause the linker to be invoked. If the linker phase is successful, the executable image is created in a file named “Hello”
  - $./Hello
    - Cause the shell to create a new process executing the image contained in the file “Hello”

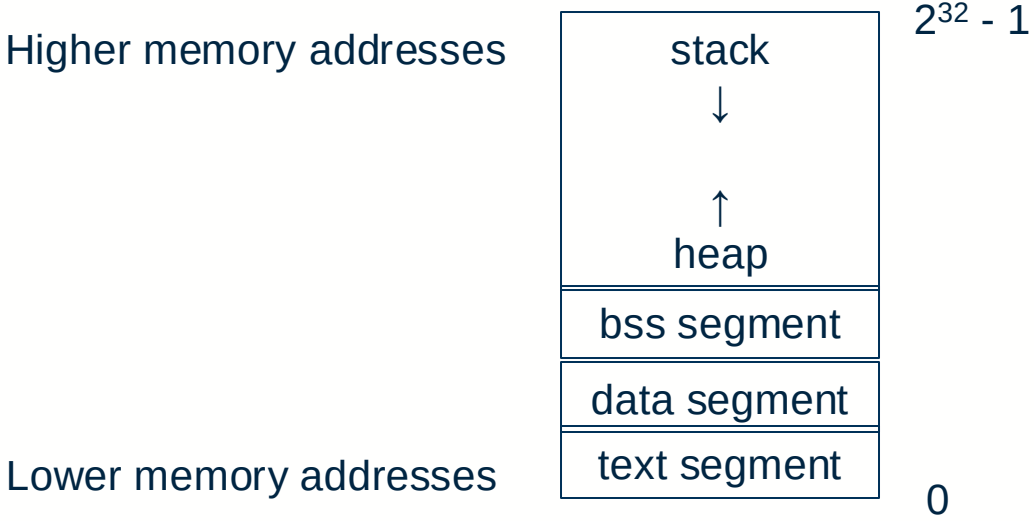
University of Glasgow School of Computing Science

Memory loading after ECLC



University of Glasgow School of Computing Science

Memory layout of a C program process (32 bit machine)



Memory layout of a C program process (32 bit machine)

- Text segment : the area in memory where instructions that the processor understands, and which resulted from the compiler's actions, are stored; that memory area is read-only
- Data segment : the area in memory where any **initialized** data in your program is stored; when the process starts to execute, the variables will have their initialized values. E.g. global and static variables that have assigned value such as constant
- Bss (Block started by symbol)segment : the area in memory where any **uninitialized** variables in your program are stored. Data in this segment is initialized to arithmetic 0 before the program starts executing
  - Uninitialized data starts at the end of the data segment
  - Contains all global or static variables that are initialized to zero
  - Do not have explicit initialization in source code

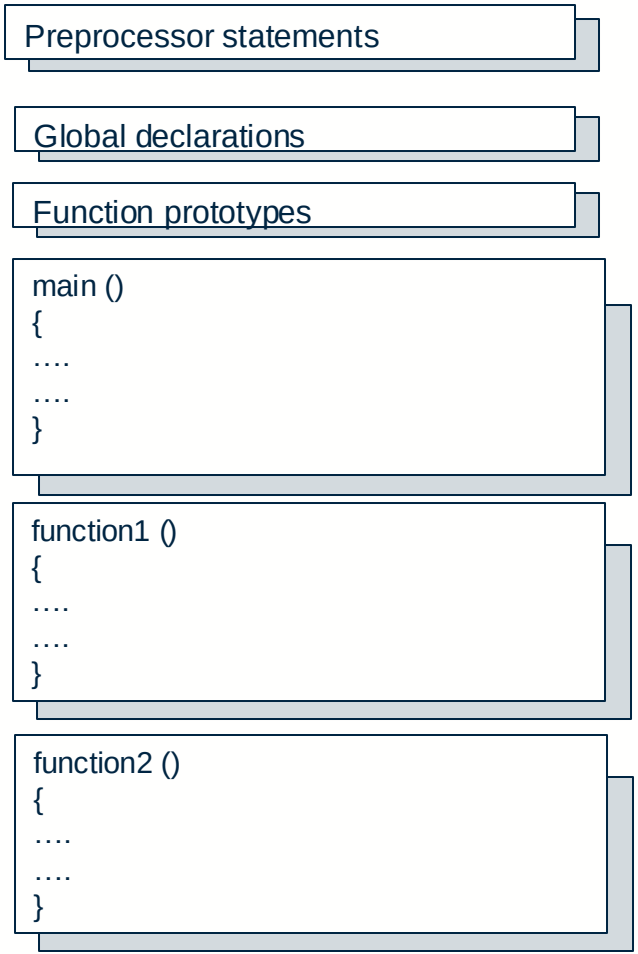
Memory layout of a C program process (32 bit machine)

- Heap : memory that is dynamically allocated, starts at the next address after the bss segment and grows upward in memory. Heap area is managed by malloc, realloc and free. It is shared by all shared libraries and dynamically loaded modules in a process
- Stack : where procedure call frames and subroutine linkage is maintained, as well as providing the storage for all automatic variables, grows downward in memory starting at the highest legal virtual address

University of Glasgow School of Computing Science

Structure of C program

- Most C programs contain these basic elements

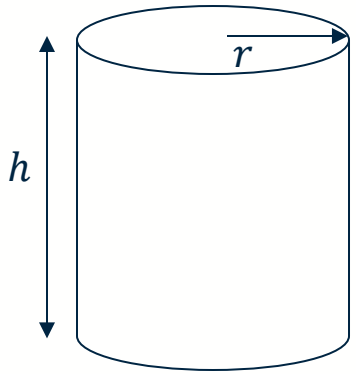


University of Glasgow School of Computing Science

A simple C program

- Preprocessor statements
  - Represents instructions to the C preprocessor. The C preprocessor is a program that analyzes source code before passing it on to the compiler. All preprocessor statements begin with the # symbol
- Global declarations
  - Define global variables. These variables are accessible to all parts of a C program
- Function Prototypes
  - Also known as function declarations. They are statements that provide the compiler with information regarding the type of value returned by a function also the type and number of function arguments
- Functions
  - Groups of statements or instructions that the computer can execute. Main() is also considered as a function

University of Glasgow School of Computing Science



volume= $\pi r^2 h$

Surface area = $2\pi rh + 2\pi r^2$

```c
/*************************************************************/
/*          cylinder.c                                       */
/*          compute the volume and surface area of a cylinder */
/************************************************************* */
#include <stdio.h>
#define PI 3.141592654

void main (void);
void main (void)
{
    float radius,height,volume,surface_area;
    /* print heading */
    printf("\n Cylinder.c");
    printf("\n computes volume and surface area of a cylinder.");

    /* read in radius and height */
    printf("\n\n Enter raduius of cylinder: ");
    scanf("%f",&radius);
    printf(" Enter height of cylinder: ");
    scanf("%f", &height);

    /* compute volume and surface area */
    volume=PI*radius*radius*height;
    surface_area=2*PI*radius*(radius+height);

    /*print results */
    printf("\n volume of cylinder is %10.4f", volume);
    printf("\n surface area of cylinder is %10.4f \n\n", surface_area);
}
```

University of Glasgow
School of Computing Science

A simple C program

```
Chees-MacBook-Pro-2:example cheekiatseowprof$ gcc -o cylinder cylinder.c
Chees-MacBook-Pro-2:example cheekiatseowprof$ ./cylinder

 Cylinder.c
 computes volume and surface area of a cylinder.

 Enter raduius of cylinder: 10
 Enter height of cylinder: 20

 volume of cylinder is  6283.1855
 surface area of cylinder is  1884.9556
```

```
/*************************************************************/
/*          cylinder.c                                       */
/*          compute the volume and surface area of a cylinder */
/************************************************************* */
#include <stdio.h>
#define PI 3.141592654

void main (void);
void main (void)
{
    float radius,height,volume,surface_area;
    /* print heading */
    printf("\n Cylinder.c");
    printf("\n computes volume and surface area of a cylinder.");

    /* read in radius and height */
    printf("\n\n Enter raduius of cylinder: ");
    scanf("%f",&radius);
    printf(" Enter height of cylinder: ");
    scanf("%f", &height);

    /* compute volume and surface area */
    volume=PI*radius*radius*height;
    surface_area=2*PI*radius*(radius+height);

    /*print results */
    printf("\n volume of cylinder is %10.4f", volume);
    printf("\n surface area of cylinder is %10.4f \n\n", surface_area);
}
```

comments

preprocessor statements

function prototype

function header

begin body of a function main ()

variable declarations

call printf() library function

call scanf() library function

assignment statements

end body of a function main ()

A simple C program (cylinder.c)

- Comments
  - All comments begin /* and end with */. Comments are remarks which help to clarify a program. Comments are ignored by the compiler
- Preprocessor statements
  - **#include <stdio.h>** is a preprocessor statement. The C preprocessor is a program that read the source code and modifies it before passing it on to the C compiler. The preprocessor directive **include** just tells the preprocessor to read the contents of the file **stdio.h** into the program. The file **stdio.h** contains information needed by the input and output functions that a part of standard C library.
  - # **define PI 3.141592654** is another preprocessor statement. It defines a symbolic constant **PI** representing the constant π

A simple C program (cylinder.c)

- Function Prototypes
  - **void main (void) ;** tells the compiler to expect a function called **main ()** , does not return a value (therefor call **void**) and also it does not have any argument.
  - All C statements end with **;** but preprocessor directives do not end with **;** since they are not C statement but rather instructions to the preprocessor
- Function header
  - **void main (void)** marks the beginning of a function **main ().** It specifies name of the function, type of value returned by the function and any information needed by the function to the compiler.
  - Parentheses **{ }** indicates beginning and end of the function body which contains the statements that make up the function

University of Glasgow School of Computing Science

A simple C program (cylinder.c)

- Global declarations
  - **Float radius, height , volume, surface area;** is a declaration statement that tells the compiler that we will be using 4 variables and these variables will be used for storing floating point numbers.
  - Compiler executes this statement and allocates memory storage for these four variables in the Bss segment since these variables have not been assigned any values.
- Functions
  - **printf("\n cylinder.c")** is a function call of the library function **printf()** which is a standard C library that comes with the C compiler. It asks the library function **printf** () to print string of character
  - **scanf("%f",&radius)** is a function call of the the library function **scanf()** which is a standard C library that comes with the C compiler. It asks the library function **scanf** () to read in a floating point value and stores this value in the variable **radius**

University of Glasgow School of Computing Science

A simple C program (cylinder.c)

- Assignment
  - **volume=PI*radius*radius*height;**

    surface_area=2*PI*radius*(radius+height;

    are two arithmetic assignment statement that computes the volume and surface area of cylinder and assign to the variables **volume** and **surface_area** respectively.

University of Glasgow School of Computing Science

Functions

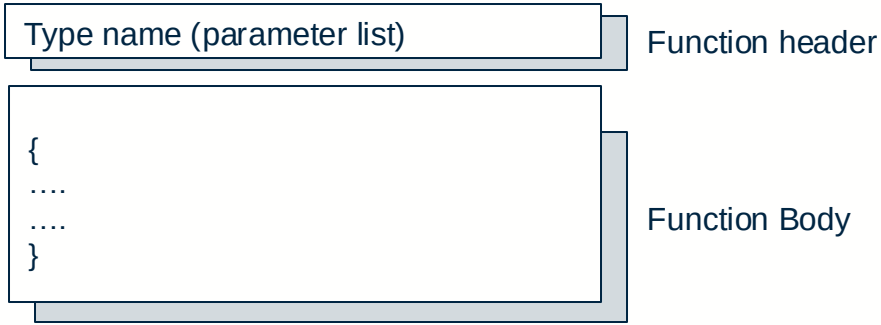
- All C function consists of a function header and a function body
- Function header format

int square (int a)

Type of value returned

function name

parameter list



- e.g. **void main (void)** , **main ()**, **float square_root(float x)**
- All C program must have a function called **main ()** as control is passed from operating system (OS) to **main ()** and program execution begins with the first statement in **main().** Control is passed back to the OS after main() program finished.

Statements

- A C statement consists of keywords, variables, function names and operators. A statement always end with semicolon ;

```
float x;                                /* declaration */
float y;                                /* declaration */
float z;                                /* declaration */
x=10;                                   /* assignment */
y=10;                                   /* assignment */
z=x+y;                                  /* arithmetic assignment */
printf("\n volume =%f", volume);        /* function call */
```

- Common type of statements but not limited to
  - Declaration statements
  - Assignment statements

University of Glasgow School of Computing Science

Statements

- Declaration statements
  - All C programs work with data
    - constant
      - Symbolic name that has values of data remains fixed throughout execution of a program. Compiler will assign the symbolic name a memory allocation at **data segment**
    - Variables
      - Symbolic name that we can assign a range of values. Change during program execution. Compiler will assign the symbolic name a memory allocation at **bss segment**
      - Unlike python or any other language , all variables in C program must be declared before use. (***Why?)***
      - For example

float x

Compiler will executes above statement and allocates a storage location in memory named **x** with storage space that accommodates float type

Statements

x ?

- what is the value of x during compile and execution?

- **float** are C language keywords. All C keywords are in lowercase. They cannot be used as variable names or function names. C language keywords are

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

Statements

- Assignment statement
  - used to assign values to variables. The assignment statement consists of a variable name followed by an equal symbol (=) and the value to be assigned to the variable.

```
x=10;                    /* assignment */
y=10;                    /* assignment */
z=x+y;                   /* arithmetic assignment */
```

C Preprocessor

- C language has a built in pre-processor C program that examines your source code before passing it on to the compiler.
- Preprocessor task
  - Changes the source code and add new sources code based on directive contained in your program
  - Provide useful feature to provide a means to modify C program in a number of way before a actual compilation take places.
  - Always included with C system and C compiler automatically call the preprocessor prior to compiling your program
  - Example : string replacement, macro expansion, file inclusion, and conditional compilation
  - All preprocessor statements begin with # character and do not have a semicolon at end of statement

University of Glasgow School of Computing Science

C Preprocessor

- **#define** Directive
  - Handling macro substitutions. A macro substitution is simply the replacement of a string of characters with another

```
#define token_string replacement_string
```

  - During compilation, once pre-processor encounters this statement, it replaces every subsequent occurrence of token_string with replacement_string
  - Primary use is to assign symbolic names to program constants especially hardware specific values such as machine address. These values can be changed for each different hardware platform
  - e.g **#define PI 3.141592654**, **#define Message "An invalid value was entered" , #define Error_Flag True**

C Preprocessor

- **#include** directive
  - used to cause the contents of the named file to be placed in the source program at the location where the **#include** statement resides

```
#include “filename”
```

  - e.g. #include “myfile.h” causes the contents of myfile.h to be included into the program and the statements in myfile.h are thus treated as if they had been typed into the program at that point.
  - Include files are also called header file. Its common use is to place all definitions and constants in such separate header file and then include these in one or more program files to ensure all programs received the same values.
  - Header files are contain structure definitions, variable declarations, function prototypes ,all of which maybe common to various modules in the program (portability?)

University of Glasgow School of Computing Science

C Preprocessor

- **#include** directive

```
#include <filename>
```

  - This **#include ”filename”** directive will search file in a prearranged list of directories while the **#include <filename>** directive will cause compiler to search in current directory first.
  - **#include <filename>** directive is part of compiler services to provide a set of header files. These files contain definitions for many of the system-dependent variables and constants. They also contains function prototypes for library functions such as **stdio.h** (stands for standard input and output) which contains system independent information needed by the input and output library functions such as **printf ()** and **scanf ().**

#include <stdio.h>

#include <math.h>

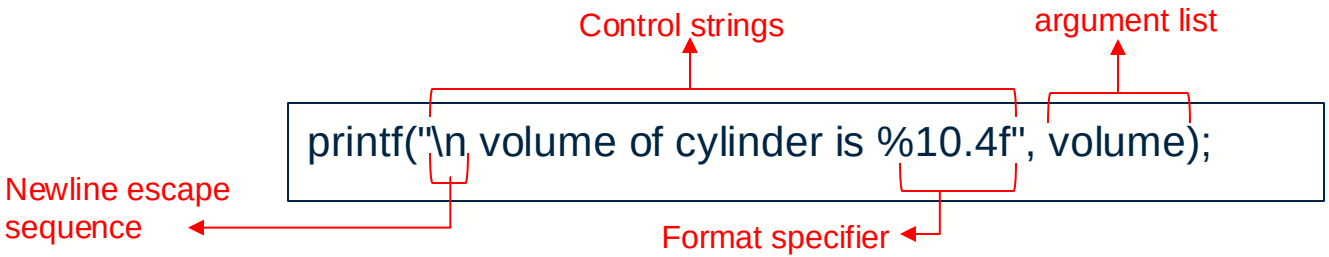
University of Glasgow School of Computing Science

Print () Function

- **#printf** function call

```
printf(“control string”, argument list);
```

  - Part of standard C library to prints the message in the control strings on the standard output screen. When we use this function that is not within the program, the compiler will make a note of this when it compiles the program and leaves a message for the linker. When the linker links the program, it searches for the code for the function in the standard C library or in any other libraries that we specify and links this code with our program



University of Glasgow School of Computing Science

Print () Function

- Control string consists of two type of items :
  - Characters to be displayed on the screen
  - Format specifiers
    - define the way in which items in the argument list are to be displayed
    - Beings with a percent sign % and is followed by the conversion character which is one character to indicate type of data to be printed. Precision can be specified

Format specifier	Data type representation
%d	print as a decimal integer
%6d	print as a decimal integer at least 6 characters wide with leading blanks
%06d	print as a decimal integer at least 6 characters wide with leading blanks
%f	print as a floating point number
%6f	print as a floating point number at least 6 characters wide
%.2f	print as a floating point number with 2 digits after the decimal point
%6.2f	print as a floating point number at least 6 characters wide with 2 digits after the decimal point
%c	print as a character
%s	print as a string
%x	print as a hexadecimal integer

```
/*print results */
printf("\n volume of cylinder is %10.4f", volume);
printf("\n surface area of cylinder is %10.4f \n\n", surface_area);
```

```
Chees-MacBook-Pro-2:example cheekiatseowprof$ gcc -o cylinder cylinder.c
Chees-MacBook-Pro-2:example cheekiatseowprof$ ./cylinder

 Cylinder.c
 computes volume and surface area of a cylinder.

 Enter raduius of cylinder: 10
 Enter height of cylinder: 20

 volume of cylinder is  6283.1855
 surface area of cylinder is  1884.9556
```

University of Glasgow School of Computing Science

Print () Function

- Escape Sequences are used to represent non printing characters, Non printing character us used to perform special functions such as newline, backslash

Escape Sequences	Characters
\a	bell
\b	backspace
\f	formfeed
\n	newline
\r	carriage return
\t	Horizontal tab
\v	Vertical tab
\:	Single quote
\”	Double quote
\\	backslash
\ddd	Octal notation
\xddd	Hexadecimal notation

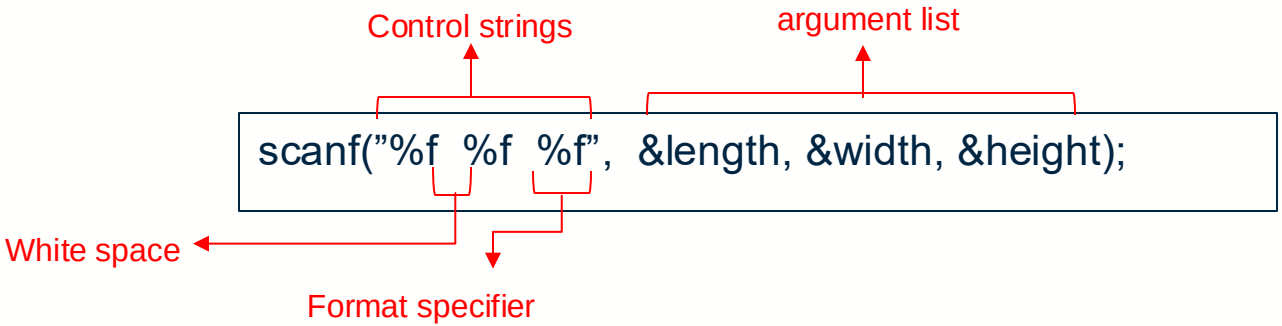
University of Glasgow School of Computing Science

Scanf () Function

- **#scanf()** function call

```
scanf("control string", argument list);
```

  - Part of standard C library to provide a general purpose input function.



University of Glasgow School of Computing Science

Scanf () Function

- In normal operation besides pointer concept, we must provide **scanf()** with the address of the variables defined in the argument list. Done by preceding the variables with ampersand **(&). &** is the address of the operator and by passing scanf () the address of the variable, we are providing scanf() with a way to access the variable and change its value

&length – memory address of variable length

&width – memory address of variable width

&height – memory address of variable height

Variable	contents	memory address
length	12	2000
width	22	2004
height	16	2008
	…	


University of Glasgow
Next Lecture
Lecture 2 : Problem Solving and Modeling
#UofGWorldChangers
@UofGlasgow