UNIVERSITI TEKNOLOGI MALAYSIA

www.utm.my

Week 4

Introduction to C

UNIVERSITI
TEKNOLOGI
MALAYSIA
www.utm.my

4.1 Parts of a C Program

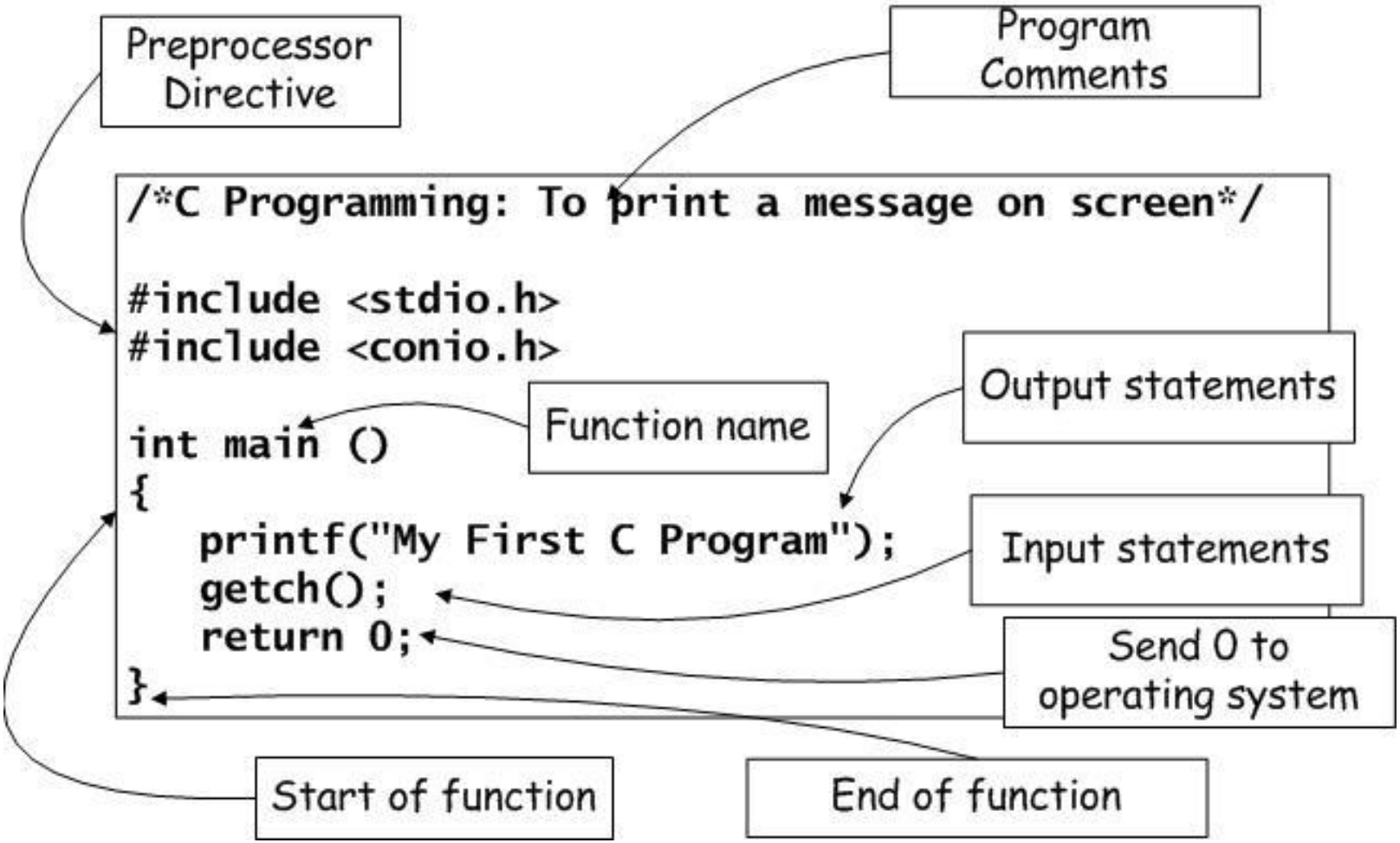
A Simple of C Program

```
/*C Programming: To print a message on screen*/

#include <stdio.h>
#include <conio.h>

void main ()
{
   printf("My First C Program\n");
   getch();
}
```

- Output:

```
My First C Program
```

A Simple of C Program



Special Characters

Character	Name	Meaning
/*	Slash star	Beginning of a comment
/**/	Star slash	End of a comment
#	Pound sign	Beginning of preprocessor directive
< >	Open/close brackets	Enclose filename in #include
( )	Open/close parentheses	Used when naming a function
{ }	Open/close brace	Encloses a group of statements
" "	Open/close quotation marks	Encloses string of characters
;	Semicolon	End of a programming statement

Preprocessor Directives

- Begin with #
- Instruct compiler to perform some transformation to file before compiling
- Example: #include <stdio.h>
  - add the *header* file stdio.h to this file
  - .h for header file
  - stdio.h defines useful input/output functions

Functions

- Consists of *header* and *body*
  - header: void main ()
  - body: contained between { and }
    - starts with location declarations
    - followed by series of statements
- More than one function may be defined
- Functions are *called* (invoked) - more later

Main Function

- Every program has one function **main**
- Header for main: void main ()
- Program is the sequence of statements between the { } following main
- Statements are executed one at a time from the one immediately following to main to the one before the }

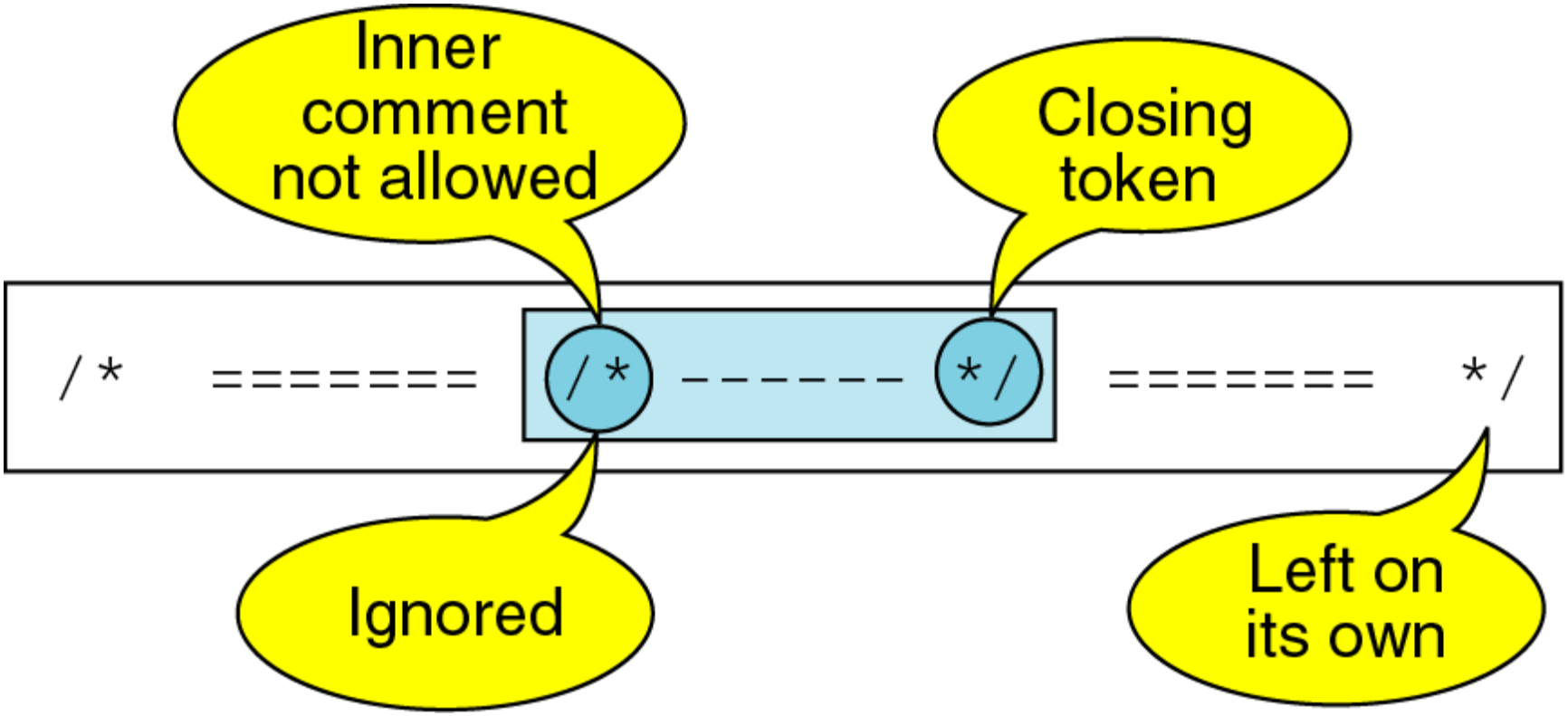
Comments

- Text between /* and */
- Used to “document” the code for the human reader
- Ignored by compiler (not part of program)
- Have to be careful
  - comments may cover multiple lines
  - ends as soon as */ encountered (so no internal comments - /* An /* internal */ comment */)

Comment Example

```
#include <stdio.h>

/* This comment covers
 * multiple lines
 * in the program.
 */

int main () /* The main header */ {
   /* No local declarations */

   printf("Too many comments\n");
} /* end of main */
```

Inner Comment



UNIVERSITI
TEKNOLOGI
MALAYSIA
www.utm.my

4.2 Tracing a simple C program

Another simple C program

```
/*The greeting program. This program demonstrates
      some of the components of a simple C program.
     Written by:  your name here
     Date       :  date program written
*/
#include <stdio.h>

int main () {
   printf("Hello world!\n");
   return 0;
}
```

Another simple C program

```
/*The greeting program. This program demonstrates
      some of the components of a simple C program.
     Written by:  your name here
     Date       :  date program written
*/

#include <stdio.h>

int main () {
   printf("Hello worl
   return 0;
}
```

A **comment** is any text between /* and */

Use comments liberally to explain your program and all its parts.

The compiler ignores all comments.

Another simple C program: Comments

- Comments in C may span several lines.

```
/* this
   is

   one

   comment */

/* this is
                another comment
*/
```

Another simple C program: Comments

- Suggestion: Line up comment delimiters vertically and use symbols such as asterisks to make your program more readable.
- Examples:

```
/* This function reads a sequence of temperatures and
 * computes the average.
 */


/***************************************
 * This program simulates a simple calculator.  *
 * It reads two numbers and an operation        *
 * (add, subtract, multiply or divide) and then   *
 * computes and prints the result.                *
 ***************************************/
```

Another simple C program

#include
means "read in
this file, too" or inserts
the contents of another
file into the program

stdio.h is the library that provides standard input/output functions (such as **printf**)

```
#include <stdio.h>

int main () {
   pri
   ret
}
```

Files ending in **.h** are called header files.

This is a preprocessor directive. All preprocessor directives start with a #

Another simple C program

```
/*The greeting program. This program demonstrates
      some of the components of a simple C
   program.
      Written by:  your name here
      Date       :  date
*/

#include <stdio.h>

int main () {
   printf("Hello world!
   return 0;
}
```

Program execution always begins in the **main** function.

All C programs must have a main function.

main() usually holds calls to other functions

Another simple C program

```
/*The greeting program. This program demonstrates
      some of the components of a simple C
```

All functions use opening and closing braces to mark the beginning and the end of the function.

```
#include <stdio.h>

int main () {

   printf("Hello world!\n");
   return 0;
}
```

The **block of statements** between these curly braces is called the **body** of the function.

Another simple C program

Function = a **block of statements** with a given **name**.

world!" on the screen.

#includ

This is the definition of a function called main, which contains two statements.

```
int main () {
   printf("Hello world!\n");
   return 0;
}
```

main() is invoked (called) automatically when a program begins execution. Other functions can be **called** from inside main()

Another simple C program: Statements

- A statement is the basic building block of a program. It usually translates to one or more machine instructions.
- All statements end in semi-colons ;
- The main() function shown in the example has two statements:

```
printf("Hello world!\n");
```

and

```
return 0;
```

Another simple C program: Functions

- A function is a block of statements with a given name, which perform a well-defined operation.

- A function has zero or more input arguments and zero or one output values.

Another simple C program

```
/*   The greeting program. This program
  demonstrates some of the components of a
#
int main () {
  printf("Hello world!\n");
  return 0;
}
```

This statement calls the printf() library function to **print** formatted text that we specify. The input argument is enclosed in parentheses. It specifies the text we want to print as well as the formatting that arranges the printed text.

ALL statements end with a semicolon!

Another simple C program: printf()

- `printf("Hello world!\n");`

The text that will be printed on the screen is
Hello world!

`\n` means move on to the next line. It is called a **format control string**. We will learn more about that later.

This statement will print Hello world! and move the cursor to the next line.

Another simple C program

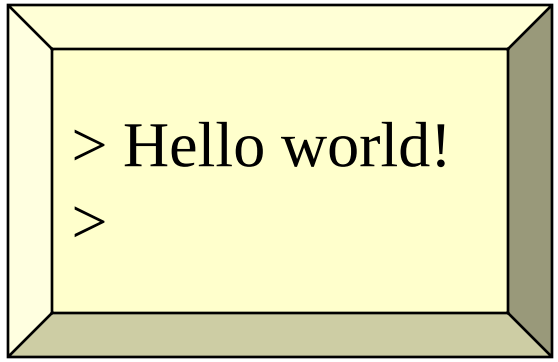
```
/*   The greeting program. This program
  demonstrates some of the components of a
  simple C program.
     Written by:  your name here
     Date:        date program written
*/
int main () {
  printf("Hello world!\n");
  return 0;
}
```

return 0; means
"…the program terminated normally (or successfully)".
More about return statements later.

ALL statements end with a semicolon!

Another simple C program: the output

```
/*The greeting program. This program demonstrates
      some of the components of a simple C program.
     Written by:  your name here
     Date       :  date program written
*/

#include <stdio.h>

int main () {
   printf("Hello world!\n");
   return 0;
}
```



Another simple C program

- C is case sensitive.
  - **printf()** is NOT the same as **Printf()**.
  - All C commands (functions) are lowercase.

- To make your program more readable:
  - Always write comments.
  - Indent your code

UNIVERSITI
TEKNOLOGI
MALAYSIA
www.utm.my

4.3 What Is a Program Made Of?

What Is a Program Made Of?

- Common elements in programming languages:
  - Key Words
  - Programmer-Defined Identifiers
  - Operators
  - Punctuation
  - Syntax

Key Words

- Also known as reserved words
- Have a special meaning in C
- Can not be used for any other purpose

Key Words

```
/*C Programming: To print a message on screen*/

#include <stdio.h>
#include <conio.h>

int main (){
   int thisYear, birthyear, age;
   thisYear=2010;
   birthyear=1980;

   age =  thisYear-birthyear;
   printf("My First C Program\n");
   printf("After I'm %d years old", age);
   getch();
   return 0;
}
```

Programmer-Defined Identifiers

- Names made up by the programmer
- Not part of the C language
- Used to represent various things: variables (memory locations), functions, etc.

Programmer-Defined Identifiers

```
/*C Programming: To print a message on screen*/

#include <stdio.h>
#include <conio.h>

int main (){
    int thisYear, birthyear, age;
    thisYear=2010;
    birthyear=1980;

    age = thisYear - birthyear;
    printf("My First C Program\n");
    printf("After I'm %d years old", age);
    getch();
    return 0;
}
```

Operators

- Used to perform operations on data
- Many types of operators:
  - Arithmetic - ex: `+, -, *, /`
  - Assignment – ex: `=`

Operators

```
/*C Programming: To print a message on screen*/

#include <stdio.h>
#include <conio.h>

int main (){
    int thisYear, birthyear, age;
    thisYear=2010;
    birthyear=1980;

    age = thisYear - birthyear;
    printf("My First C Program\n");
    printf("After I'm %d years old", age);
    getch();
    return 0;
}
```

Punctuation

- Characters that mark the end of a statement, or that separate items in a list

Punctuation

```
/*C Programming: To print a message on screen*/

#include <stdio.h>
#include <conio.h>

int main (){
    int thisYear, birthyear, age;
    thisYear=2010;
    birthyear=1980;

    age =  thisYear - birthyear;
    printf("My First C Program\n");
    printf("After I'm %d years old", age);
    getch();
    return 0;
}
```

Syntax

- The rules of grammar that must be followed when writing a program
- Controls the use of key words, operators, programmer-defined symbols, and punctuation

Exercise Week4_1

- Refer to Program 1.4 in pg. 14
- Identify the following elements
  - Key Words
  - Programmer-Defined Identifiers
  - Operators
  - Punctuation
  - Syntax

Exercise Week4_1

```
1   //Program 1.4
2   //Nama pengaturcara: Norazah Yusof
3   #include <stdio.h>
4   #include <conio>
5   int main (void)
6   {
7      int workDays;
8      float workHours, payRate, weeklyPay;
9      workDays = 5;
10     workHours = 6.5;
11     payRate = 20.50;
12     weeklyPay = workDays  * workHours * payRate;
13     printf ("Weekly Pay = %f", weeklyPay);
14     printf ("\n");
15        getch();
16     return 0;
17  }
```

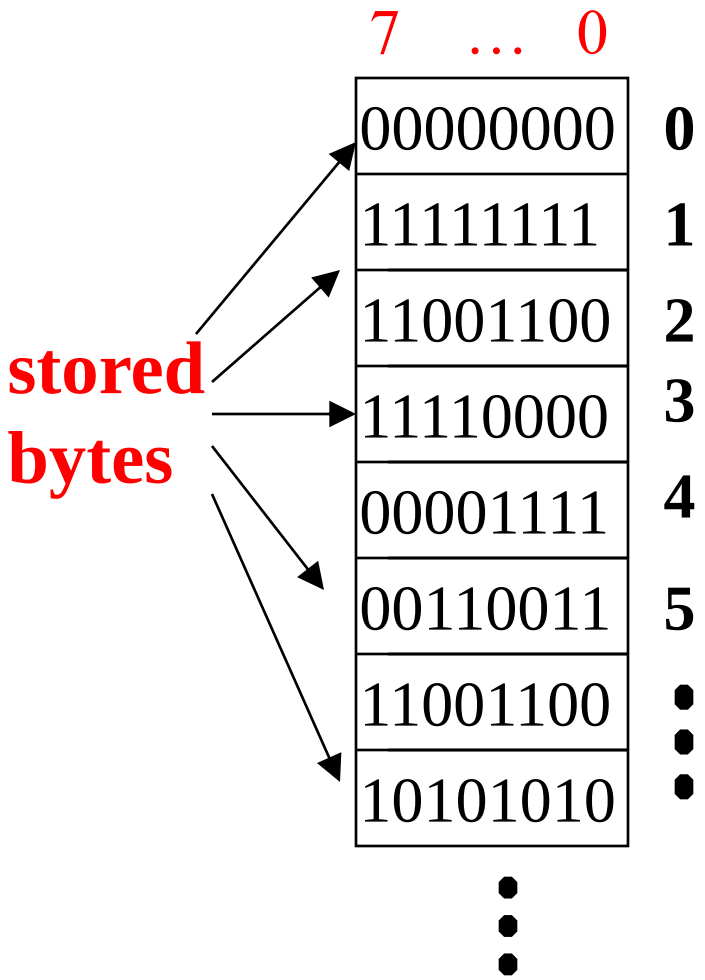
UNIVERSITI TEKNOLOGI MALAYSIA

www.utm.my

4.4 Memory & Data Storage

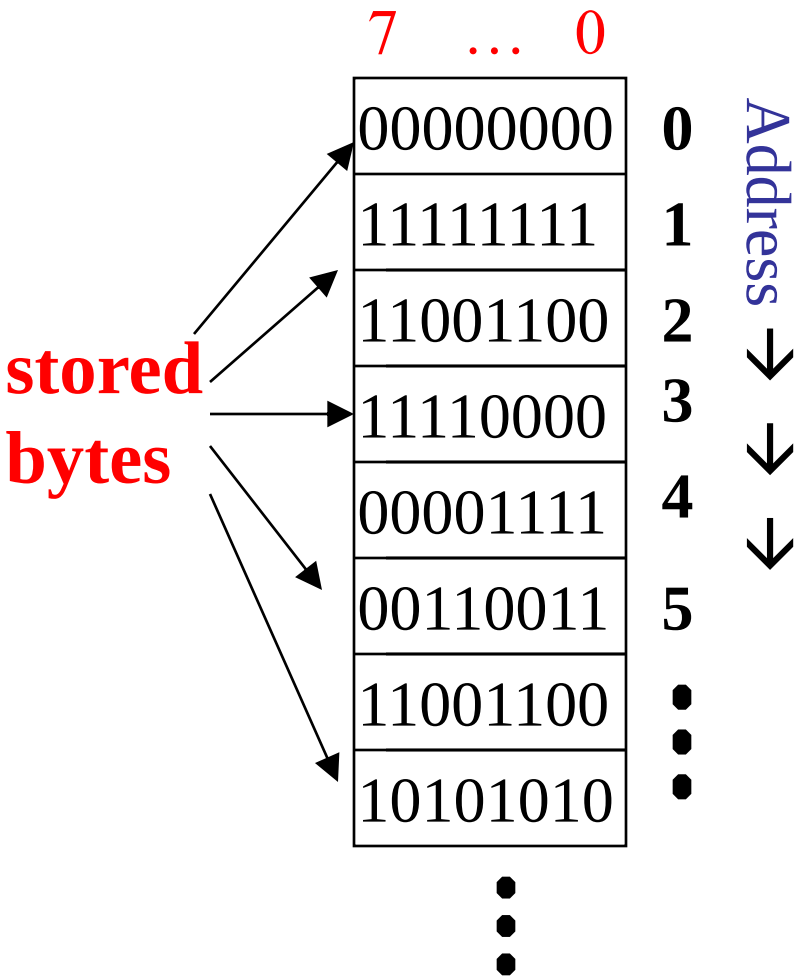
Bits, bytes and memory

- Our computer's memory can be seen as a sequence of cells.
- Each cell is 8 bits (one byte) large.
- Data is stored by setting these bits to 1s and 0s.
- 8bits = 1 byte
- 2 bytes = 1 word
- 2 words = 1 long word



Bits, bytes and memory (cont.)

- Each cell has an address.
- We don't need to know (or care to know) what this address is.
- Our system uses the address to locate the data stored there.
- Max value stored in and address:
  - **255_{10} (11111111_2)**
- Min value stored in and address:
  - **0_{10} (00000000_2)**



UNIVERSITI TEKNOLOGI MALAYSIA

www.utm.my

4.5 Variables

Variables

- We need to be able to store data in memory, during the execution of our program.
- We also need to be able to access and even modify this data.
- Solution : variables
- A variable is a reserved location in memory that
  - has a name
  - has an associated type (for example, integer)
  - holds data which can be modified

Variables

- In order to use a variable in our program we must first declare it.
- HOW?
  - A *declaration statement* has the format:

```
type  variable_name  ;
```

  - type : what kind of data will be stored in that location (integer? character? floating point?)
  - variable_name : what is the name of the variable?
  - semi-colon : this is a statement!

Variables

Variables declaration

```
                                    miles to kilometers.

                                                    /* printf, scanf definitions  */
#define KMS_PER_MILE 1.609                          /* conversion constant        */

int main(void)
{
    double miles,    /* input - distance in miles.          */
           kms;      /* output - distance in kilometers     */
    /* Get the distance in miles. */
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);
    /* Convert the distance to kilometers. */
    kms = KMS_PER_MILE * miles;
    /* Display the distance in kilometers. */
    printf("That equals %f kilometers.\n", kms);
    return (0);
}
```

Variable types

- There are four basic data types in C

Type	C keyword to use:
Integer	`int`
Floating point	`float`
	`double`
Character	`char`

Variable names

- Selecting good variable names is **important** for **program readability**.

- A variable name must be descriptive of the data that will be stored in the variable.

- It must not be too long.
- It must not be a single character

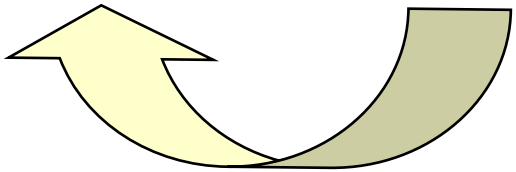
Variable values

- After a variable has been declared, its memory location contains randomly set bits. In other words, it does not contain any valid data.
- The value stored in a variable must be initialized before we can use it in any computations.
- There are two ways to initialize a variable:
  - by assigning a value using an assignment statement
  - by reading its value from the keyboard (more on that later)

Variable values

- The basic syntax of an assignment statement is

```
variable  =  value  ;
```



assign the value on the right hand side to the variable on the left hand side

- Example

```
int num_students;
num_students = 22;
```

UNIVERSITI TEKNOLOGI MALAYSIA

www.utm.my

5.5 Variables and Literals

Variables and Literals

- Variable: a storage location in memory
  - Has a name and a type of data it can hold
  - Must be defined before it can be used:

```
int item;
```

Literals

- Literal: a value that is written into a program's code.

  `"hello, there"` (string literal)

  `12` (integer literal)

Literals

- Are used to initialize a variable.
- Example:

```
char keypressed;
keypressed = 'y'; /* 'y' is a character literal */
```

- Example:

```
double pi;
pi = 3.14; /* 3.14 is a floating-point literal.
              Floating-point literals are of
              type double by default */
```

- Example:

```
int index;
index = 17; /* 17 is an integer literal */
```

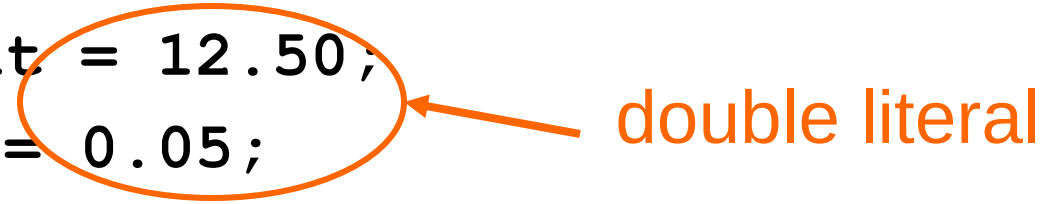
Example of literal usage

```
/* sample program that demonstrates
   variable declaration and
   initialization. */

#include <stdio.h>

int main () {
  int num_students;
  num_students = 22;
  return 0;
}
```

22 is an integer literal

Example of literal usage

```
/* sample program that demonstrates variable
   declaration and initialization. */

#include <stdio.h>

int main () {
   double rate, amount;   /* declare two
                             double variables */
   amount = 12.50;
   rate = 0.05;
   return 0;
}
```



Example of literal usage

```
/*The greeting program. This program demonstrates
      some of the components of a simple C program.
     Written by:  your name here
     Date       :  date program written
*/

#include <stdio.h>

int main () {
   printf("Hello world!\n");
   return 0;
}
```

This is also a string literal

UNIVERSITI TEKNOLOGI MALAYSIA
www.utm.my

5.6 Identifiers

Identifiers

- An identifier is a programmer-defined name for some part of a program: variables, functions, constants and label.
- 2 type of identifier:
  1. Standard identifier – used in C library e.g. `printf` and `scanf`
  2. User defined identifier – 3 objectives of the identifier are variable, constant & function

User Defined Identifiers

- Identifiers rules:
  - Consists a combination of letters, digits, underscore (_)
  - Cannot begin with a digit.
  - Upper- and lowercase characters are distinct
  - Only the first 31 characters of a variable name are significant. The rest are ignored.
  - A C-keywords word cannot be used

C Keywords

You cannot use any of the C key words as an identifier. These words have reserved meaning.

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

Valid and Invalid Identifiers

IDENTIFIER	VALID?	REASON IF INVALID
`totalSales`	Yes	
`total_Sales`	Yes	
`total.Sales`	No	Cannot contain .
`4thQtrSales`	No	Cannot begin with digit
`totalSale$`	No	Cannot contain $

Exercise Week4_2

VALID	INVALID	REASON IF INVALID
`utm`	`2utm`	
`_bek`	`meow?`	
`rekod_201`	`rekod-301`	
`ifi`	`if`	

Exercise Week4_3

- Identify 7 errors in the following program.

```
#include <stdio.h>
#include <conio.h>

int main (){
    int nama_yang_tersangat_panjang_jenis_int;
    float nama_yang_tersangat_panjang_jenis_float;
    const float kadar = 25.23, goto=1.3;
    float pinjambank, pinjamkawan, samanpolis,hutang;
    char kod;
    int bil_guli = 5.0;

    Hutang= pinjambank*kadar+pinjambak+pinjamkawan+samanpolis;
    kadar=20.1; nama_yang_tersangat_panjang_jenis_int =80000;
    kod = 66;
}
```

User Defined Identifiers : Constants

- Constant = named memory location that holds a non-changeable value
  - MUST be initialized with a value
  - Can not be modified after initialization

- 2 ways to declare constant
  - Using type qualifier **`const`**
  - Using **`#define`** preprocessor directive

User Defined Identifiers : Constants

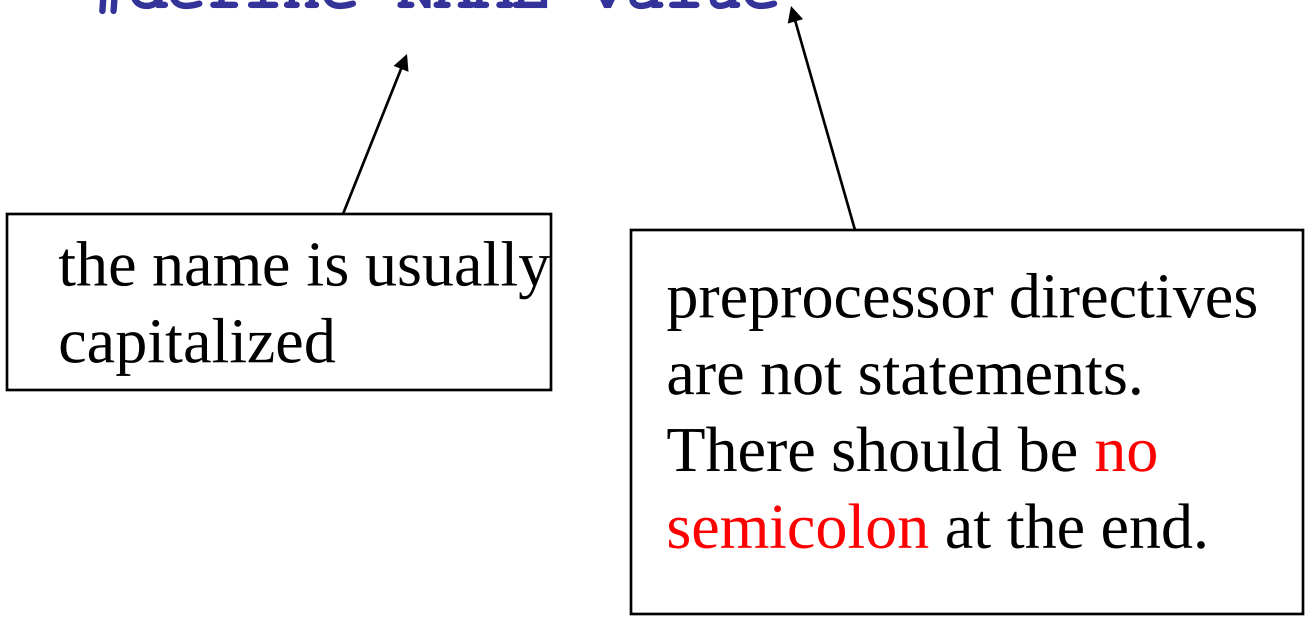
- Declared using type qualifier **const**

```
int main () {
      const double pi = 3.14;
      double area, radius;
      radius = 12;
      area = pi * radius * radius;
      return 0;
}
```

`#define` preprocessor directive

- Syntax:

```
#define NAME value
```



`#define` preprocessor directive

- Example use:

```
#define PI 3.14
int main () {
      double area, radius;
      radius = 12.1;
      area = PI * radius * radius;
      return 0;
}
```

Instructs preprocessor to replace all occurrences of PI with 3.14

NOTE : no semicolon after preprocessor directives!

#define preprocessor directive

- Advantage : if you need to change the value, you only have to do it once

if I want to change the value of PI to 3.14159, I only have to do it here and the preprocessor will take care of the rest.

```
#define PI 3.14
int main () {
      double area, radius;
      double circumference
      radius = 12.6;
      area = PI * radius * radius;
      circumference = 2 * PI * radius;
      return 0;
}
```

UNIVERSITI TEKNOLOGI MALAYSIA

www.utm.my

2.6 Integer Data Types

Integer Data Types

- Integer variables can hold whole numbers such as 12, 7, and -99.

Table 2-6 Integer Data Types, Sizes, and Ranges

Data Type	Size	Range
`short`	2 bytes	–32,768 to +32,767
`unsigned short`	2 bytes	0 to +65,535
`int`	4 bytes	–2,147,483,648 to +2,147,483,647
`unsigned int`	4 bytes	0 to 4,294,967,295
`long`	4 bytes	–2,147,483,648 to +2,147,483,647
`unsigned long`	4 bytes	0 to 4,294,967,295

Defining Variables

- Variables of the same type can be defined
  - On separate lines:

```
int length;
int width;
unsigned int area;
```

  - On the same line:

```
int length, width;
unsigned int area;
```

- Variables of different types must be in different definitions

Integer Literals

- An integer literal is an integer value that is typed into a program's code. For example:

```
itemsOrdered = 15;
```

In this code, 15 is an integer literal.

Integer Literals

- Integer literals are stored in memory as `int`s by default
- To store an integer constant in a long memory location, put '`L`' at the end of the number: `1234L`
- Constants that begin with '`0`' (zero) are base 8: `075`
- Constants that begin with '`0x`' are base 16: `0x75A`

Exercise Week4_4

- Refer to Algorithm 2.4 in Lab 2 pg. 18
- Convert the algorithm into correct C++ code to calculate the total of three integer numbers.

UNIVERSITI
TEKNOLOGI
MALAYSIA
www.utm.my

2.7 The `char` Data Type

The `char` Data Type

- Used to hold characters or very small integer values
- Usually 1 byte (8 bits)of memory
- Numeric value of character from the character set is stored in memory:

CODE:

```
char letter;
letter = 'C';
```

MEMORY:

letter

Character Constants

- Character literals must be enclosed in single quote marks. Examples:

```
'A', '\n'
```

- IMPORTANT : Note that the value of a character is enclosed in single quotes.
- Each character is essentially "encoded" as an integer.

Character Constants

- A computer normally stores characters using the ASCII code (American Standard Code for Information Exchange)
  - ASCII is used to represent
    - the characters A to Z (both upper and lower case)
    - the digits 0 to 9
    - special characters (e.g. @, <, etc)
    - special control codes
  - For example,
    - the character 'A' is represented by the code 65
    - the character '1' is represented by the code 49
  - IMPORTANT: the integer 1, the character '1' and the ASCII code 1 represent three different things!

Character Constants

- **`char`** (continued)
  - Part of ASCII Table
  - The decimal code will be converted into binary (stored as a integer byte)
  - E.g. A
    - = 65 (decimal)
    - = 0100 0001 (binary)

Decimal (code)	Symbol
65	A
66	B
67	C
68	D
69	E
:	:
97	a
98	b
99	c
100	d
101	e

Character Constants

- Escape sequences – enable the use of special symbols.

Char	Special symbols
`'\n'`	New line
`'\t'`	horizontal tab
`'\v'`	vertical tab
`'\r'`	carriage return
`'\b'`	backspace
`'\f'`	formfeed
`'\\'`	Backslash (\)
`'\x41'`	Hexa 0x41
`'\101'`	Octal 101
`'\0'`	null

Character Strings

- A series of characters in consecutive memory locations:

  `"Hello"`

- Stored with the null terminator, `\0`, at the end:

H	e	l	l	o	\0

- Comprised of the characters between the `" "`

UNIVERSITI TEKNOLOGI MALAYSIA

www.utm.my

2.8 Floating-Point Data Types

Floating-Point Data Types

- The floating-point data types are:
  `float`
  `double`
  `long double`

- They can hold real numbers such as:

```
12.45       -3.8
```

- Stored in a form similar to scientific notation

- All floating-point numbers are signed

Floating-Point Data Types

Table 2-8 Floating Point Data Types on PCs

Data Type	Key Word	Description
Single precision	`float`	4 bytes. Numbers between ±3.4E-38 and ±3.4E38
Double precision	`double`	8 bytes. Numbers between ±1.7E-308 and ±1.7E308
Long double precision	`long double*`	8 bytes. Numbers between ±1.7E-308 and ±1.7E308

Floating-point Literals

- Can be represented in
  - Fixed point (decimal) notation:

    `31.4159` `0.0000625`
  - E notation:

    `3.14159E1` `6.25e-5`
- Are `double` by default
- Can be forced to be float (`3.14159f`) or long double (`0.0000625L`)

Exercise Week4_5

- Refer to Lab 4 Exercise 2 no. 1 in pg. 46
- Solve the problem.

UNIVERSITI TEKNOLOGI MALAYSIA
www.utm.my

2.11 Variable Assignments and Initialization

Variable Assignments and Initialization

- An assignment statement uses the = operator to store a value in a variable.

```
item = 12;
```

- This statement assigns the value 12 to the `item` variable.

Assignment

- The variable receiving the value must appear on the left side of the = operator.
- This will NOT work:

```
// ERROR!
12 = item;
```

Variable Initialization

- To initialize a variable means to assign it a value when it is defined:

```
int length = 12;
```

- Can initialize some or all variables:

```
int length = 12, width = 5, area;
```

Variable Initialization - example

```
/*This program shows variable initialization*/
#include <stdio.h>
#include <conio.h>

int main (){
   int thisYear=2010, birthyear=1980, age;
   age =  thisYear-birthyear;
   printf("My First C Program. ");
   printf("After I'm %d years old.", age);
   getch();
   return 0;
}
```

- Output:

```
My First C Program
After I'm 30 years old
```

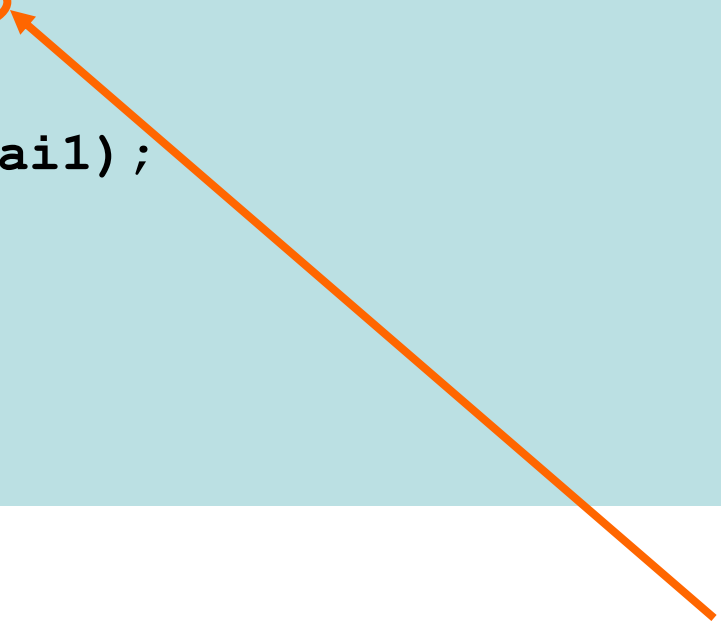
2.12 Scope

Scope

- The scope of a variable: the part of the program in which the variable can be accessed
- A variable cannot be used before it is defined

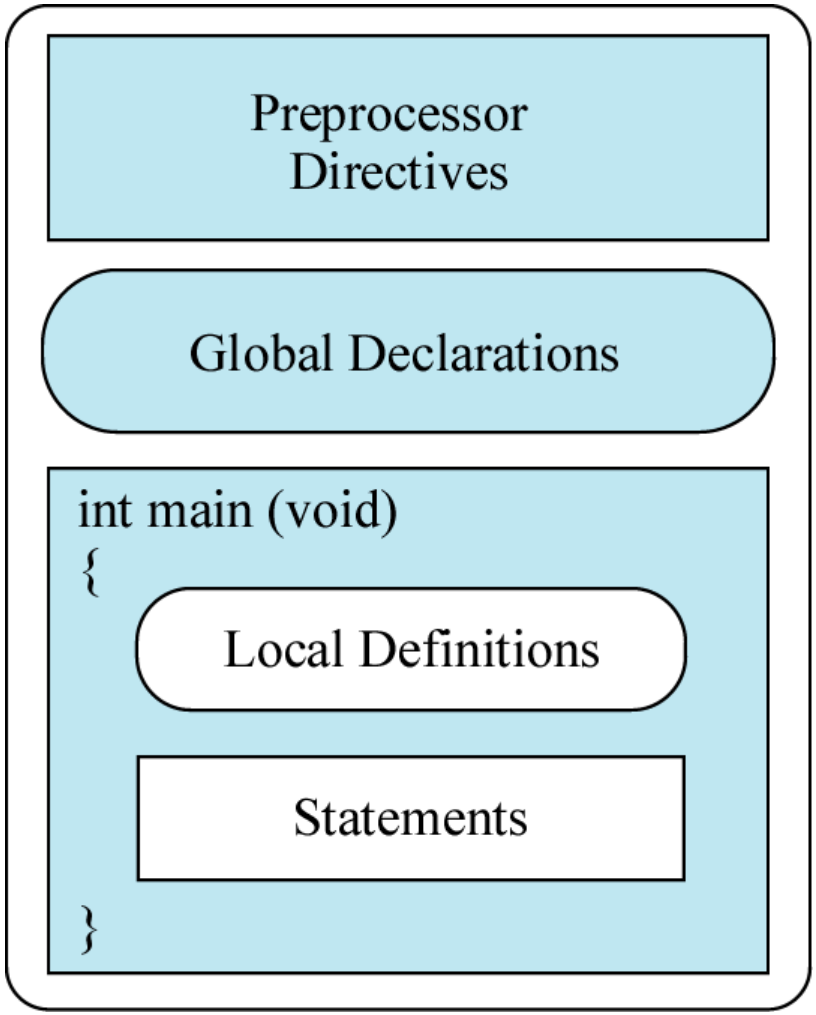
Scope - example

```
int main (){
  nilai1 = 4;
   int nilai1 ;
   printf("%d", nilai1);
   getch();
   return 0;
}
```



Error! **`nilai1`** not defined yet

Structure of C Program



Structure of C Program

```
/*C Programming: To print a message on screen*/

#include <stdio.h>
#include <conio.h>

int main (){
    int thisYear, birthyear, age;
    thisYear=2010;
    birthyear=1980;

    age =  thisYear-birthyear;
    printf("My First C Program\n");
    printf("After I'm %d years old", age);
    getch();
    return 0;
}
```

Preprocessor Directive

No Global Variable

Local Variables

Statements

- Output:

```
My First C Program
After I'm 30 years old
```

Program Execution

- Global declarations set up
- Function *main* executed
  - local declarations set up
  - each statement in statement section executed
    - executed in order (first to last)
    - changes made by one statement affect later statements

Exercise Week4_6

- Refer to Exercise 3 no. 1-6 in pg. 47
- Solve the problem.

UNIVERSITI
TEKNOLOGI
MALAYSIA
www.utm.my

2.15 Programming Style

Programming Style

- The visual organization of the source code
- Includes the use of spaces, tabs, and blank lines
- Does not affect the syntax of the program
- Affects the readability of the source code

Programming Style

Common elements to improve readability:

- Braces { } aligned vertically
- Indentation of statements within a set of braces
- Blank lines between declaration and other statements
- Long statements wrapped over multiple lines with aligned operators

C doesn't care much about spaces

```
#include <stdio.h>/* My first C program which prints Hello World */
int main(){printf("Hello World!\n");return 0;}
```

```
#include <stdio.h>
/* My first
C program
which prints
Hello World */
int
main
(
)
{
printf
(
"Hello World!\n"
)
;
return
0
```

Both of these programs are exactly the same as the original as far as your compiler is concerned.

Note that words have to be kept together and so do things in quotes.

Indentation

```
int main () {
      printf("Hello world!\n");
      return 0;
}
```

- As you can see, the two statements in the body of main() do not line up with the rest of the code.
- This is called **indentation**.
- Orderly indentation is very important; it makes your code readable.
  - The C compiler ignores white space
  - The Borland C++ editor will help you ("smart indenting“)

Good Style

- Format 1

```
#include <stdio>
main ( )
{
    int umur;
    umur = 125;
    printf("umur saya %d tahun", umur);
}
```

- Format 2

```
#include <stdio> main ( ) {    int umur;
    umur = 125;  printf("umur saya %d tahun", umur); }
```

Good Style

- Format 3

```
#include <stdio>
main ( ) {

    int umur;
    umur = 125;
    printf("umur saya %d tahun", umur);
}
```

- Format 4

```
#include <stdio>
main ( ) { int
          umur;
         umur
       = 125;
     printf("
            umur saya %d tahun", umur
              );
}
```

UNIVERSITI TEKNOLOGI MALAYSIA
www.utm.my

Thank You

Q & A