Advanced Programming (C & C++)

Fundamental of C

Mahboob Ali

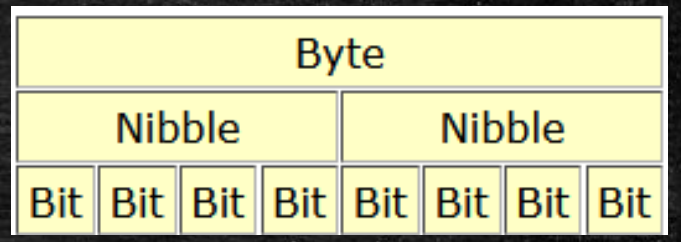
Fundamental Units

Bits

- The most fundamental unit of a modern computer is the binary digit or *bit*.
- A bit is either on or off.
- One (1) represents on, while zero (0) represents off.
- Since bits are too numerous to handle individually, modern computers transfer and handle information in larger units.
- As programmers, we define some of those units.

Bytes

- The fundamental addressable unit of RAM is the *byte*.
- One byte consists of 2 nibbles.
- Each *nibble* consists of 4 bits
- One byte can store any one of 256 (2^8) possible values in the form of a bit string:
- The bit strings are on the left.
- The equivalent decimal values are on the right.
- Note that our counting system starts from 0, not from 1.




```
00000000 <- 0
00000001 <- 1
00000010 <- 2
00000011 <- 3
00000100 <- 4
...
00111000 <- 56
...
11111111 <- 255
```

Words

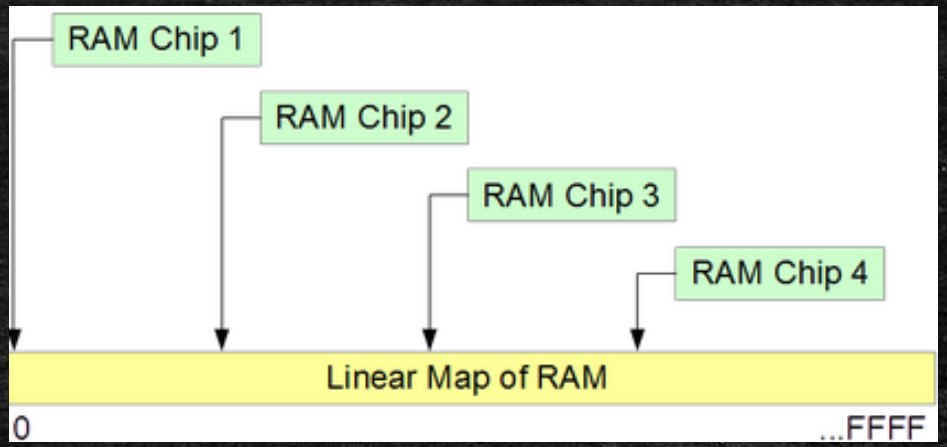
- We call the natural size of the execution environment a *word*.
- A word consists of an integral number of bytes and is typically the size of the CPU's general registers.
- Word size may vary from CPU to CPU.
- On a 16-bit CPU, a word consists of 2 bytes.
- On a Pentium 4 CPU, the general registers contain 32 bits and a word consists of 4 bytes.
- On an Itanium 2 CPU, the general registers contain 64 bits, but a word still consists of 4 bytes.

Hexadecimal

- The decimal system is not the most convenient numbering system for organizing information.
- The hexadecimal system (base 16) is much more convenient.
- Two hexadecimal digits holds the information stored in one byte.
- Each digit holds 4 bits of information. The digit symbols in the hexadecimal number system are {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}.
- The characters A through F denote the values that correspond to the decimal values 10 through 15, respectively. We use the 0x prefix to identify a number as hexadecimal (rather than decimal - base 10).
- For example, the hexadecimal value 0x5C is equivalent to the 8-bit value 010111002, which is equivalent to the decimal value 92.

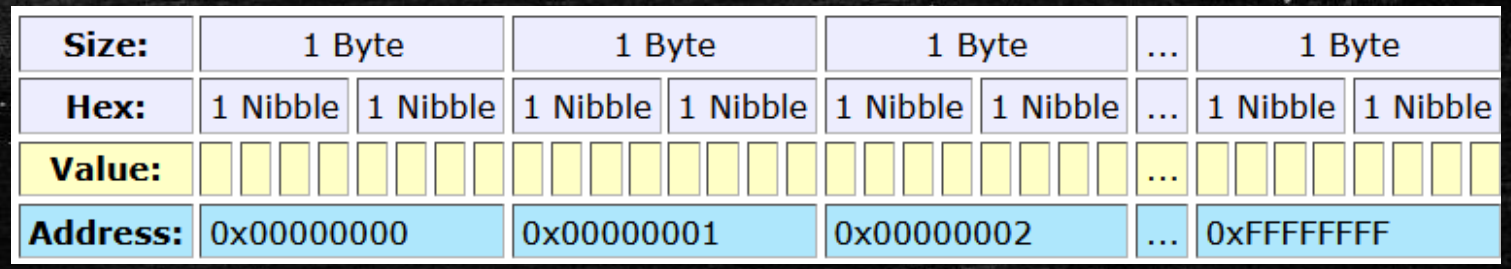
Memory Model

- The memory model for organizing information stored in RAM is linear.
- Any byte in memory is accessible through a map that treats each actual physical memory location as a position in a continuous sequence of locations aligned next to one another.



Addresses

- Each byte of RAM has a unique address. Addressing starts at zero, is sequential, and ends at the address equal to the size of RAM less 1 unit.
- For example, 4 Gigabytes of RAM
  - consists of 32 (= 4 * 8) Gigabits
  - starts at a low address of oxoooooooo
  - ends at a high address of oxFFFFFFFF
- Note that each byte, and not each bit, has its own address.
- We say that RAM is byte-addressable.



Sets of bytes

- The abbreviations for sets of bytes are:

- Kilo or k (=1024): 1 Kilobyte = 1024 bytes ~ 10^3 bytes
- Mega or M (=1024k): 1 Megabyte = 1024 * 1024 bytes ~ 10^6 bytes
- Giga or G (=1024M): 1 Gigabyte = 1024 * 1024 * 1024 bytes ~ 10^9 bytes
- Tera or T (=1024G): 1 Terabyte = 1024 * 1024 * 1024 * 1024 bytes ~ 10^{12} bytes
- Peta or P (=1024T): 1 Petabyte = 1024 * 1024 * 1024 * 1024 * 1024 bytes ~ 10^{15} bytes
- Exa or E (=1024P): 1 Exabyte = 1024 * 1024 * 1024 * 1024 * 1024 * 1024 bytes ~ 10^{18} bytes

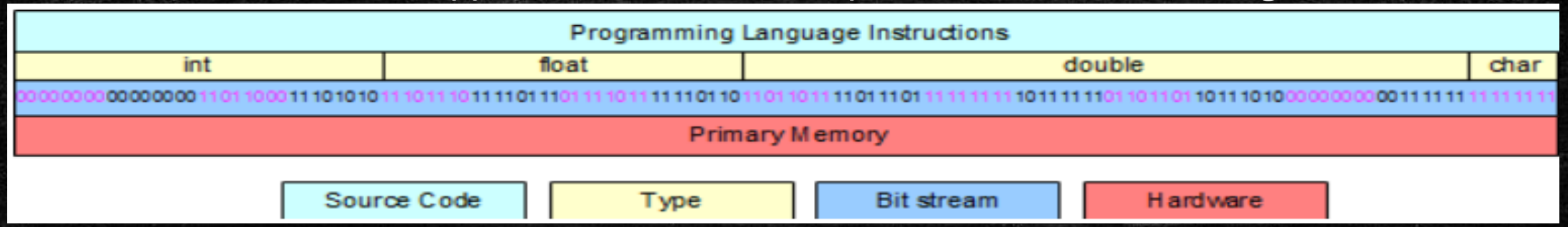
- Note that the multiplying factor is 1024, not 1000. 1024 bytes is 2^{10} bytes, which is approximately 10^3 bytes.

Segmentation Faults

- The information stored in RAM consists of information that serves different purposes.
- We expect to read and write data, but not to execute it.
- We expect to execute program instructions but not to write them.
- So, certain architectures assign the data read and write permissions, while assigning instructions read and execute permissions.
- Such permission system helps trap errors while a program is executing.
- An attempt to execute data or to overwrite an instruction reports an error.
- Clearly, the access has been to the wrong segment. We call such errors a *segmentation faults*.

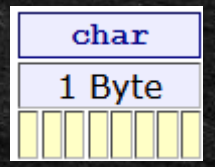
Types

- A typed programming language uses a type system to interpret the bit streams in memory.
- C is a typed programming language.
- A type is the rule that defines how to store values in memory and which operations are admissible on those values.
- A type defines the number of bytes available for storing values and hence the range of possible values.
- We use different types to store different information.
- The relation between types and raw memory is illustrated in the figure below.

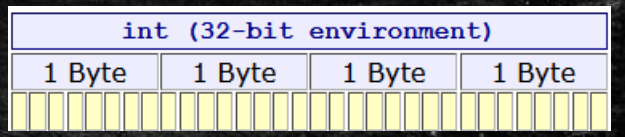


Arithmetic types

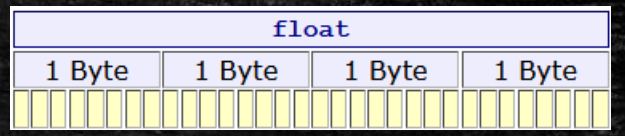
- The four most common types of the C language for performing arithmetic calculations are:
  - char
  - int
  - float
  - double
- A char occupies one byte and can store a small integer value, a single character or a single symbol:



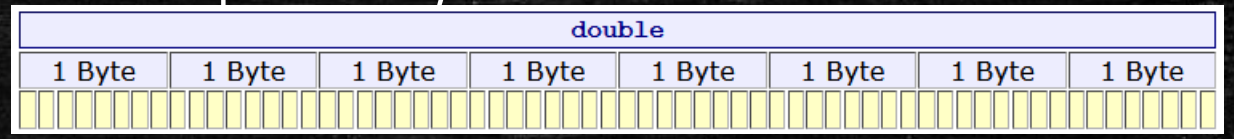
- An int occupies one word and can store an integer value. In a 32-bit environment, an int occupies 4 bytes:



- A float typically occupies 4 bytes and can store a single-precision, floating-point number:

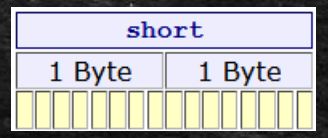


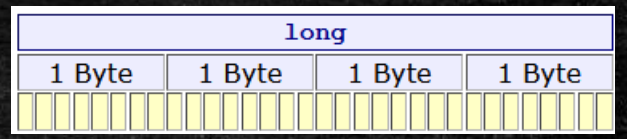
- A double typically occupies 8 bytes and can store a double-precision, floating-point number:

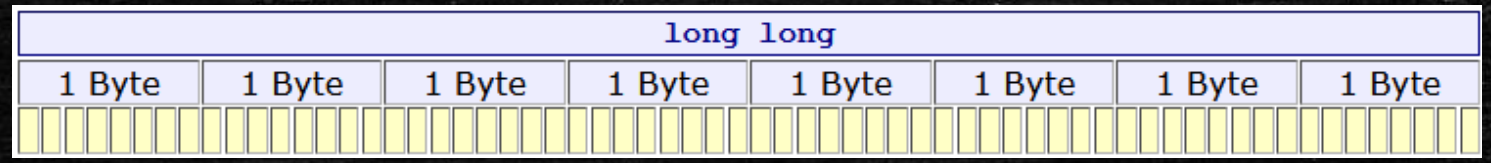


Size specifiers and their types

- Size specifiers adjust the size of the `int` and `double` types.
- Specifying the size of an int ensures that the type contains a minimum number of bits. The three specifiers are:
  - `short`
  - `long`
  - `long long`
- A `short int` (or simply, a short) contains at least 16 bits:
- A `long int` (or simply, a long) contains at least 32 bits:
- A `long long int` (or simply, a long long) contains at least 64 bits:







`const` Qualifier

- Any type can hold a constant value.
- A constant value cannot be changed.
- To qualify a type as holding a constant value we use the keyword `const`.
- A type qualified as `const` is unmodifiable.
- That is, if a program instruction attempts to modify a `const` qualified type, the compiler will report an error.

Variable declarations

- We store program data in variables A declaration associates a program variable with a type. The type identifies the properties of the variable.
- In C, a declaration takes the form

```
[const] type identifier [= initial value];
```

- We select a meaningful name for the identifier and optionally set the variable's initial value. We conclude the declaration with a semi-colon, making it a complete statement.
- For example,
  - `char  children;`
  - `int   nPages;`
  - `float cashFare;`
  - `const double pi = 3.14159265;`

Multiple declarations

- We may group the identifiers of variables that share the same type within a single declaration by separating the identifiers by commas.
- For example,
  - `char   children, digit;`
  - `int    nPages, nBooks, nRooms;`
  - `float  cashFare, height, weight;`
  - `double loan, mortgage;`

Naming Conventions

- We may select any identifier for a variable that satisfies the following naming conventions:
  - starts with a letter or an underscore (_)
  - contains any combination of letters, digits and underscores (_)
  - contains less than 32 characters (some compilers allow more, others do not)
  - is not be a C reserved word

Reserved Words

C language

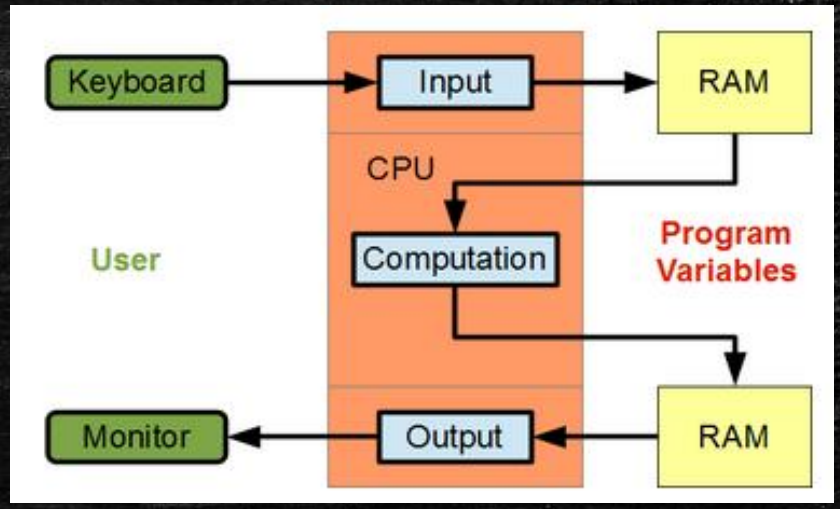
auto	_Bool	break	case
char	_Complex	const	continue
default	restrict	do	double
else	enum	extern	float
for	goto	if	_Imaginary
inline	int	long	register
return	short	signed	sizeof
static	struct	switch	typedef
union	unsigned	void	volatile
while			

C++ (due to upward compatibility)

asm	export	private	throw
bool	false	protected	true
catch	friend	public	try
class	mutable	reinterpret_cast	typeid
const_cast	namespace	static_cast	typename
delete	new	template	using
dynamic_cast	operator	this	virtual
explicit			wchar_t

Program flow

- Application programs receive input from the user, convert that input into output and display the output to the user.
- The user enters the input through a keyboard or similar device and receives converted output through a monitor or similar device.
- Program instructions store the input data in RAM, retrieve that input data from RAM, convert it using the ALU and FPA in the CPU and store the result in RAM
- A program that directs these operations consists of both program variables and program instructions that operate on data stored in those variables.



Escape sequence

- Character constants include special actions and symbols.
- We define special actions and symbols by *escape sequences*.
- The backslash (\) before each symbol identifies the symbol as part of an escape sequence:

Character	Sequence
alarm	\a
backspace	\b
form feed	\f
newline	\n
carriage return	\r
horizontal tab	\t
vertical tab	\v
backslash	\\
single quote	\'
double quote	\"
question mark	\?

String literal

- A string literal is a sequence of characters enclosed within a pair of double quotes. For example,

```
"This is C\n"
```

- The \n character constant adds a new line to the end of the string.

Simple Input

- The `scanf(...)` instruction accepts data from the user (that is, the standard input device) and stores that data in memory at the address of the specified program variable. The instruction takes the form

```
scanf(format, address);
```

- This statement calls the `scanf()` procedure, which performs the input operation.
- We say that format and address are the arguments in our call to `scanf()`.

format

- *format* is a string literal that describes how to convert the text entered by the user into data stored in memory.
- *format* contains the conversion specifier for translating the input characters.
- Conversion specifiers begin with a % symbol and identify the type of the destination variable.

Specifier	Input Text is a	Destination Type
%c	character	char
%d	decimal	int, short
%f	floating-point	float
%lf	floating-point	double

address

- *address* contains the address of the destination variable.
- We use the prefix & to refer to the 'address of' of a variable.

```
&radius
```

Accept radius of the circle

```c
#include <stdio.h>                          // for printf, scanf
 int main(void)
 {
     const float pi = 3.14159f;    // pi is a constant float
     float radius;                 // radius is a float

     printf("Enter radius : ");    // prompt user for radius input
     scanf("%f", &radius);         // accept radius value from user

         // ... completed in coming slides

         return 0;
 }
```

Accept radius of the circle

- The argument in the call to scanf() is the address of radius, not the value of radius.
- Coding the value radius as the argument is likely to generate a run-time error

```
scanf("%f", radius); // *** ERROR possibly SEGMENTATION FAULT ***
```

- Missing `&` in the call to `scanf()` is a common mistake for beginners and does not necessarily produce a compiler error or warning.
- Some compilers accept options (such as -W) to produce warnings, which may identify such errors.

Simple output

- The `printf(...)` instruction reports the value of a variable or expression to the user (that is, copies the value to the standard output device).

```
printf(format, expression);
```

- This statement calls the `printf()` procedure, which performs the operation.
- We say that format and expression are arguments in our call to `printf()`

format

- *format* is a string literal describing how to convert data stored in memory into text readable by the user.
- *format* contains the conversion specifier and any characters to be output directly.
- The conversion specifier begins with a % symbol and identifies the type of the source variable.

- The default number of decimal places displayed by `%f` and `%lf` is 6.
- To display two decimal places, we write `%.2f` or `%.2lf`.

Specifier	Output as a	Use With Type	Most Common
%c	character	char	*
%d	decimal	char, int	*
%f	floating-point	float	*
%lf	floating-point	double	*

expression

- *expression* is a placeholder for the source variable.
- The printf() procedure copies the variable and converts it into the output text.

Exercises

- Write a C program that prompts the user to enter the amount of cash in their pockets, accepts the user's input, and displays the amount in the format shown below.
- If the user has entered 4.52, your program displays

```
How much money do you have in your pockets ? 4.52
The amount of money in your pockets is $4.52
```

- Write a C program that calculates the area of a triangle. Your program prompts the user to enter the height and base, accepts the user's input in decimal format, multiplies the product of the height and base by 0.5, stores the area in memory and outputs the area to 2 decimal places along with its address in memory in hexadecimal format (use the %p conversion specifier to output an address).