Revision: A simple program in C

```c
/* Fig. 2.1: fig02_01.c
   A first program in C */
#include <stdio.h>

int main()
{
   printf( "Welcome to C!\n" );

   return 0;
}
```

Comments

- Text surrounded by **`/*`** and **`*/`** is ignored by computer
- Used to describe program

`#include <stdio.h>`

- Preprocessor directive - tells computer to load contents of a certain file
- **`<stdio.h>`** allows standard input/output operations

Revision: Simple program

- **`int main()`**
  - C++ programs contain one or more functions, exactly one of which must be **`main`**
  - Parenthesis used to indicate a function
  - **`int`** means that main "returns" an integer value
  - Braces indicate a block
    - The bodies of all functions must be contained in braces

Revision: Simple program

- **`printf( "Welcome to C!\n" );`**
  - Instructs computer to perform an action
    - Specifically, prints string of characters within quotes
  - Entire line called a statement
    - All statements must end with a semicolon
  - **`\`** - escape character
    - Indicates that **`printf`** should do something out of the ordinary
    - **`\n`** is the newline character

Revision: Simple program

- **`return 0;`**
  - A way to exit a function
  - **`return 0`**, in this case, means that the program terminated normally
- Right brace **`}`**
  - Indicates end of **`main`** has been reached
- Linker
  - When a function is called, linker locates it in the library
  - Inserts it into object program
  - If function name misspelled, linker will spot error because it cannot find function in library

Variables and Expressions

- Variables:

  locations in memory where a value can be stored
- Expressions:

  -represents a single data items as number or character

Character Set

- Alphabets

  A B C ....Z

  a b c ....z

- Digits

  0 1 2 3 4 5 6 7 8 9

- Special character

  , : . < > {} () [] / \ “” ‘ ! @ # $ % ^ & * + -

- White spaces

  blank space   newline   vertical tab   horizontal tab

Keywords

- Predefined by language
- Cannot be used by programmer anyway other than that specified by syntax
- Must be written in lowercase
- Examples

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

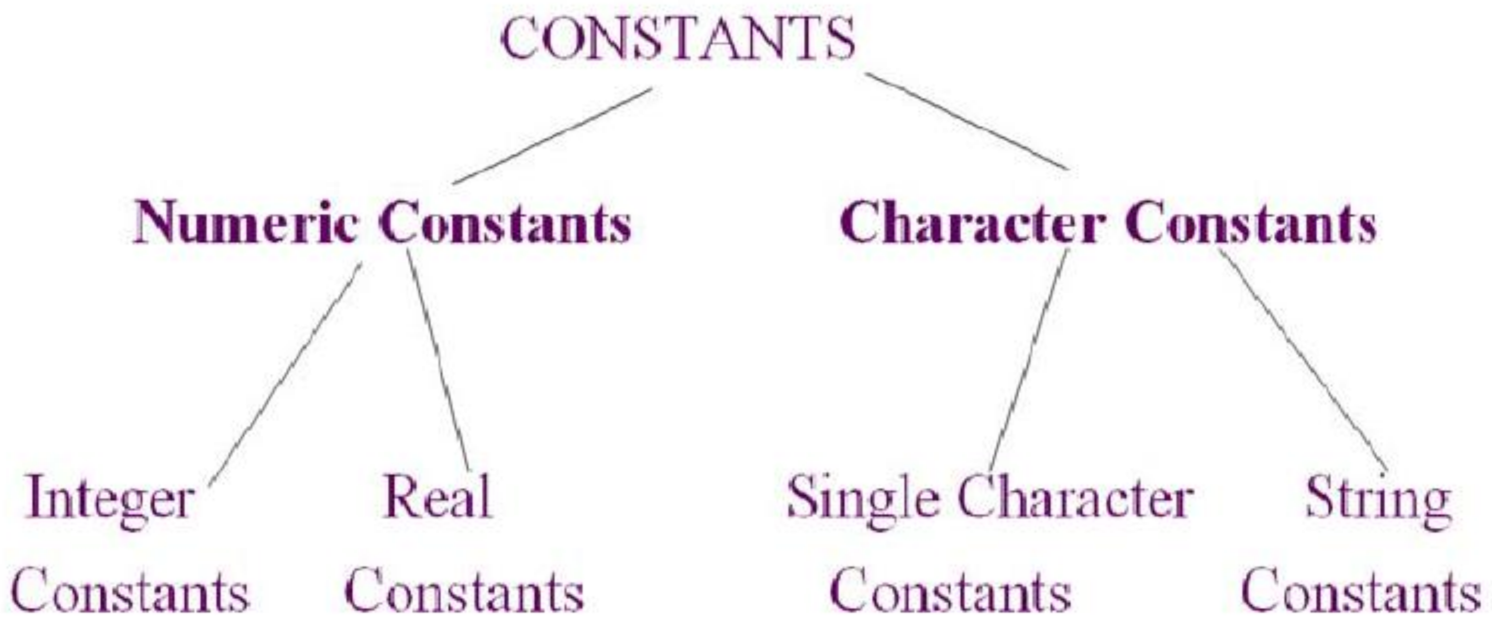
Identifiers

- Used to identify or name variables, constants, functions etc
- Identifiers can be sequence of letters followed by numbers

  x12 area   sum ram5   sita
- Identifier can start with underscore : _
- Names should not be same as keywords
- Case sensitive

Constants

- Fixed value that do not change during execution
- Classification



Constants

- Numeric constant refers to sequence of digit
  - Integer constant
    - Set of digits from 0 to 9 precede by – or +
    - eg. 12 , -45, +90 , -50
  - Real constant
    - eg. 2.5, 4.6, -9.7
- Character constant refers to sequence of character
  - Single character constant
    - Contain single character
    - eg. ‘a’, ‘o’, ‘5’, ‘.’
  - String constant
    - Sequence of character
    - eg. “ram”, “456”, “s”, “8”

- Escape sequences

```
printf(“hello world.This is my first program”);
        Output: hello world.This is my first program

printf(“hello world. \nThis is my first program”);
Output:     hello world.
            This is my first program
```

- List of escape sequence

`\a`	alert (bell) character	`\\`	backslash
`\b`	backspace	`\?`	question mark
`\f`	formfeed	`\'`	single quote
`\n`	newline	`\"`	double quote
`\r`	carriage return	`\`*ooo*	octal number
`\t`	horizontal tab	`\x`*hh*	hexadecimal number
`\v`	vertical tab		

Variables

- Use to store data value
- Chosen by programmer
- Variable names
  - Variable name can be sequence of letters followed by numbers or underscore
    abc , abc23, a76bh, no90, name_ss, si_int
    3abc is not allowed
  - Variable can start with underscore : _
    _sum, _abc
  - Names should not be same as keywords
    int as variable name is not allowed
  - Case sensitive
    abc ≠ ABC ≠ Abc ≠ aBc
  - White space is not allowed
    abc 123 is not allowed, it should be abc123

Data Type

- Primary Data Types
  - int
  - char
  - float
  - double

- User Defined Data Type
- Secondary / Derived Data Type

Primary Data Type

- Integer Types
  - defined by int
  - Signed and unsigned
  - To control storage space, C has three classes of integer storage namely short int, int and long int

Name	Size
short int	1bytes (8 bits)
int	2bytes
long int	4bytes

Using variable

- Declare Variable

  int a;

- Define Variable

  a=10;

- Initialize Variable

  int a=10;

Scope of variables

```
#include<stdio.h>

int length;

main()
{
    int width;
    {
        int height;
    }

}

anotherfunction()
{
    int area;
}
```

Global Declaration Section

Global variable

Local variable

C Program: Adding two integers

```c
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

int main()
{
   int integer1, integer2, sum;        /* declaration */

   printf( "Enter first integer\n" );  /* prompt */
   scanf( "%d", &integer1 );           /* read an
   printf( "Enter second integer\n" ); /* prompt */
   scanf( "%d", &integer2 );           /* read an
   sum = integer1 + integer2;          /* assignment of
   printf( "Sum is %d\n", sum );       /* print sum */

   return 0;  /* indicate that program ended
}
```

Simple C Program: Adding Two Integers (I)

- As before
  - Comments, **`#include <stdio.h>`** and **`main`**
- **`int integer1, integer2, sum;`**
  - Declaration of variables
    - Variables: locations in memory where a value can be stored
  - **`int`** means the variables can hold integers (**`-1`**, **`3`**, **`0`**, **`47`**)
  - **`integer1`**, **`integer2`**, **`sum`** - variable names (identifiers)
    - Identifiers: consist of letters, digits (cannot begin with a digit), and underscores, case sensitive
  - Declarations appear before executable statements
    - If not, syntax (compile) error

Simple C Program: Adding Two Integers (II)

- **`scanf( "%d", &integer1 );`**
  - Obtains value from user
    - **`scanf`** uses standard input (usually keyboard)
  - This **`scanf`** has two arguments
    - **`%d`** - indicates data should be a decimal integer
    - **`&integer1`** - location in memory to store variable
    - **`&`** is confusing in beginning - just remember to include it with the variable name in **`scanf`** statements
      - It will be discussed later
  - User responds to **`scanf`** by typing in number, then pressing the enter (return) key

Simple C Program: Adding Two Integers (III)

- **`=`** (assignment operator)
  - Assigns value to a variable
  - Binary operator (has two operands)

    `sum = variable1 + variable2;`

    `sum` gets **`variable1 + variable2;`**
  - Variable receiving value on left
- **`printf( "Sum is %d\n", sum );`**
  - Similar to **`scanf`** - **`%d`** means decimal integer will be printed
    - **`sum`** specifies what integer will be printed
  - Calculations can be performed inside **`printf`** statements

    ```
    printf( "Sum is %d\n", integer1 + integer2 );
    ```

Memory Concepts

- Variables
  - Variable names correspond to *locations* in the computer's memory.
  - Every variable has a *name,* a *type,* a *size* and a *value.*
  - Whenever a new value is placed into a variable (through **`scanf`**, for example), it
  - replaces (and destroys) previous value
  - Reading variables from memory does not change them
- A visual representation

`integer1` $\boxed{\textbf{45}}$

Arithmetic

- Arithmetic calculations are used in most programs
  - Use `*` for multiplication and `/` for division
  - Integer division truncates remainder

    `7 / 5` evaluates to **1**
  - Modulus operator returns the remainder

    `7 % 5` evaluates to **2**
- Operator precedence
  - Some arithmetic operators act before others (i.e., multiplication before addition)
    - Use parenthesis when needed
  - Example: Find the average of three variables **`a`**, **`b`** and **`c`**
    - Do not use: `a + b + c / 3`
    - Use: `(a + b + c ) / 3`

Arithmetic (II)

- Arithmetic operators:

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	`+`	*f* + *7*	`f + 7`
Subtraction	`-`	*p* – *c*	`p - c`
Multiplication	`*`	***bm***	`b * m`
Division	`/`	*x* / *y*	`x / y`
Modulus	`%`	***r mod s***	`r % s`

- Rules of operator precedence:

Operator(s)	Operation(s)	Order of evaluation (precedence)
`()`	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
`*`, `/`, or `%`	Multiplication Division Modulus	Evaluated second. If there are several, they re evaluated left to right.
`+` or `-`	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Decision Making: Equality and Relational Operators

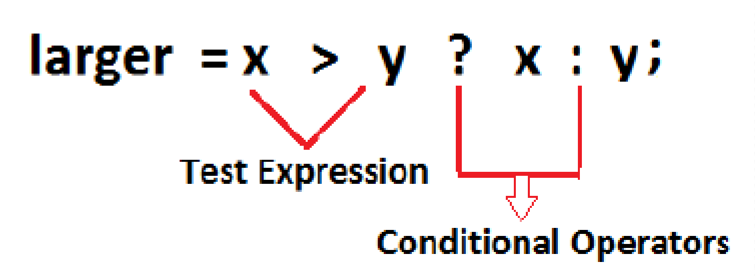
- Executable statements
  - Perform actions (calculations, input/output of data)
  - Perform decisions
    - May want to print "pass" or "fail" given the value of a test grade
- **`if`** control structure
  - Simple version in this section, more detail later
  - If a condition is true, then the body of the **`if`** statement executed
    - **`0`** is false, non-zero is true
  - Control always resumes after the **`if`** structure
- Keywords
  - Special words reserved for C
  - Cannot be used as identifiers or variable names

Conditional Operators

- Consist of two symbols : question mark (?) and colon (:) which are called ternary operators

```
exp1 ? exp2 : exp3
```

- If exp1 is true exp2 is value of expression else exp3 is value of expression
- Example



```
larger =x  >  y  ?  x : y;
```

Decision Making: Equality and Relational Operators (II)

Standard algebraic equality operator or relational operator	C++ equality or relational operator	Example of C++ condition	Meaning of C++ condition
Relational operators			
>	>	**x** > **y**	**x** is greater than **y**
<	<	**x** < **y**	**x** is less than **y**
≥	>=	**x** >= **y**	**x** is greater than or equal to **y**
≤	<=	**x** <= **y**	**x** is less than or equal to **y**
Equality operators			
=	==	**x** == **y**	**x** is equal to **y**
≠	!=	**x** != **y**	**x** is not equal to **y**

- Write a program to input two integers and display all the possible relationships between the integers input.
- Eg.

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```