Data types, variables, constants

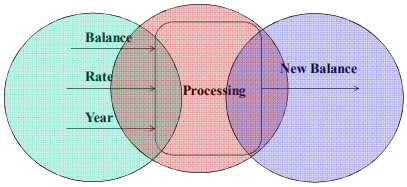
Outline

Definition: Computer Program

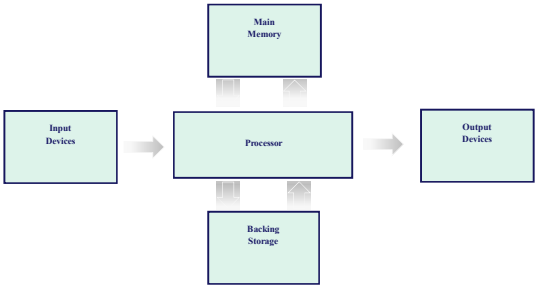
A Computer program is a sequence of instructions written to perform a specified task with a computer.

Example: Computer Program

Computer tell me what would be my balance after 1 year if my starting balance is 1,00,000 and interest rate is 10%.



A simple model of the computer



Introduction

- C programming language
  - Structured and disciplined approach to program design
  - Story → Paragraph → Sentence
  - Program → Function → Statement

A Simple C Program: Printing a Line of Text

```
1  /* Fig. 2.1: fig02_01.c
2     A first program in C */
3  #include <stdio.h>
4
5  int main()
6  {
7     printf( "Welcome to CSE 115!\n" );
8
9     return 0;
10 }
```

```
Welcome to CSE 115!
```

- 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

A Simple C Program: Printing a Line of Text (II)

- **`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

A Simple C Program: Printing a Line of Text (III)

- **`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

A Simple C Program: Printing a Line of Text (IV)

- **`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

Basic Data types

Type	Size	Range	Format Code
char	`8 bits, 1 byte`	*-128 to +127*	`"%c"`
int	`32 bits, 4 bytes`	*-2,14,74,83,648 to +2,14,74,83,647*	`"%d"` `"%i"`
float	`32 bits, 4 bytes`	*3.4E-38 to 3.4E+38*	`"%f"`
double	`64 bits, 8 bytes`	*1.7E-308 to 1.7E+308*	`"%lf"`

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`	`45`

Data input

```
int a;
scanf("%i", &a);
    or
scanf("%d", &a);
```

```
float a;
scanf("%f ", &a);
```

```
double a;
scanf("%lf ", &a);
```

```
char a;
scanf("%c", &a);
```

Output: value stored in a variable

```
int a = 10;
printf("%d", a);
```

```
float a = 10.4;
printf("%f ", a);
```

```
double a = 90.3;
printf("%lf ", a);
```

```
char a = 'b';
printf("%c", a);
```

```
char a = 'b';
printf("%d", a);
```

Little QUIZ (What is the output if user enters 5)

```
int a;
scanf("%i", &a);
printf("%d", a);
```

```
char a;
scanf("%c ", &a);
printf("%d", a);
```

Naming convention of a variable

- Capital letters A-Z, lowercase letters a-z, digits 0-9, and the underscore character
- First character must be a letter or underscore
- Usually only the first 31 characters are significant
- There can be no embedded blanks
- Keywords cannot be used as identifiers
- Identifiers are case sensitive

Key words in C

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

Same valid/invalid variable names

- OK First_tag
- NO char
- NO Price$
- NO group one
- OK average_number
- OK int_
- NO 8boys

Declaring variables

```
1  /* Fig. 2.1: fig02_01.c
2     A first program in C */
3  #include <stdio.h>
4
5  int main()
6  {
7     int interestrate;
8     char ch;
9     return 0;
10 }
```

Assigning values

- Two ways:
  - Using **scanf**

```
int c;
scanf("%d",&c);
```

  - Using **assignment operator**

```
int c;
c = 10;
```

Arithmetic

- 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	%	$r \bmod s$	`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 are evaluated left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Arithmetic (II)

- 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`

Working with int…examples

- To extract digits of a number:
  - Given 4 digit number, can you reverse it?
  - Modulus (%) and division (/) operators are good enough.
- Time difference of two cities.
  - Dhaka: 11:20
  - Mumbai: 10:50

Military time to standard time

Name Date

MILITARY TIME CHART 2

Military Time	Standard Time
00:00	12:00 midnight
01:00	1:00am
02:00	2:00am
03:00	3:00am
04:00	4:00am
05:00	5:00am
06:00	6:00am
07:00	7:00am
08:00	8:00am
09:00	9:00am
10:00	10:00am
11:00	11:00am
12:00	12:00 midday
13:00	1:00pm
14:00	2:00pm
15:00	3:00pm
16:00	4:00pm
17:00	5:00pm
18:00	6:00pm
19:00	7:00pm
20:00	8:00pm
21:00	9:00pm
22:00	10:00pm
23:00	11:00pm
00:00	12:00 midnight

Free Math Sheets, Math Games and Math Help
KYH-SALAMANDERS.COM

Increment and Decrement Operators

- Increment operator `(++)` can be used instead of `c=c+1`
- Decrement operator `(--)` can be used instead of `c=c-1`.
- Preincrement
  - Operator is used before the variable (`++c` or `--c`)
  - Variable is changed, then the expression it is in is evaluated
- Postincrement
  - Operator is used after the variable (`c++` or `c--`)
  - Expression executes, then the variable is changed

Increment and Decrement Operators (II)

- When variable not in an expression
  - Preincrementing and postincrementing have the same effect.

```
++c;
printf("%d",c);
```

and

```
c++;
 printf("%d",c);
```

have the same effect.

Increment and Decrement Operators (III)

- When variable in an expression
  - Pre-incrementing and post-incrementing DOES NOT have the same effect.
  - Preincrement updates the variable first then evaluates expression
  - Postincrement evaluates the expression first then updates the variable

If `c = 5`, then

```
printf( "%d", ++c);
```

Prints **6**

```
printf( "%d", c++);
```

Prints **5**

In either case, `c` now has the value of **6**

Assignment Operators (shorthand notations)

- Assignment operators abbreviate assignment expressions

```
c = c + 3;
```

can be abbreviated as `c += 3;` using the addition assignment operator

- Statements of the form

variable = *variable operator expression* ;

can be rewritten as

variable operator= *expression* ;

- Examples of other assignment operators:

```
d -= 4      (d = d - 4)
e *= 5      (e = e * 5)
f /= 3      (f = f / 3)
g %= 9      (g = g % 9)
```

Working with fractions: Example 1

- Sonali Bank provides interests at a certain rate to all its clients having a savings account with the bank. Write down a program that will take initial balance, and interest rates and will determine and print:
  - (1) Balance after one year
  - (2) Balance after two years
  - (3) Balance after *n* years where *n* will also be input to your program.

Working with characters

- Print ASCII values.
- Convert uppercase to lower case and vice versa
- Write down a program that will take a small letter as input and will print *n*-th letter starting from the letter given as input (wrap around). Assume that *n* is a nonnegative integer less than or equal to 26. Prompt the user to know the value of *n*.

Type conversion

- Lower to higher auto-conversion (called auto-casting)

```
int x = 9;
float y = x;   //OK no warning no error
```

- Higher to lower still auto-casting but generates warning

```
float x = 9.5;
int y = x;   //OK but generates warning but no error
Int y = (int) x // No warning called casting
```

- Work out the followings:

```
float x = 5/3;
int y = 5/3;
```

```
float x = 5.0/3;
int y = 5.0/3;
```

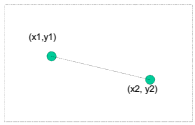
Type conversion (example)

Floor(x)	: The largest integer not exceeding x
Ceil	: The smallest integer not less than x
Round	: The nearest integer (in case of tie take greater one)

Problem Solving Methodology

1. **State the problem clearly**
2. **Describe the input/output information**
3. **Work the problem by hand, give example**
4. **Develop a solution** (Algorithm Development) **and Convert it to a program** (C program)
5. **Test the solution with a variety of data**

Working with fractions: Example 2



1. Problem statement

Compute the straight line distance between two points in a plane

2. Input/output description

Example 2 (cont'd)

3. Hand example

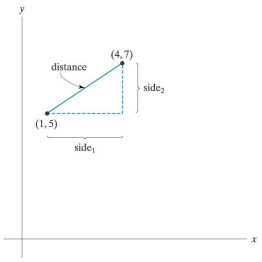
$side1 = 4 - 1 = 3$

$side2 = 7 - 5 = 2$

$distance = \sqrt{side1^2 + side2^2}$

$distance = \sqrt{3^2 + 2^2}$

$distance = \sqrt{13} = 3.61$



Example 2 (cont'd)

4. Algorithm development and coding

a. Generalize the hand solution and list/outline the necessary operations step-by-step

1) Give specific values for point1 (x1, y1) and point2 (x2, y2)
2) Compute side1=x2-x1 and side2=y2-y1
3) Compute $distance = \sqrt{side1^2 + side2^2}$
4) Print distance

b. Convert the above outlined solution to a program using any language you want (see next slide for C imp.)

Example 2 (cont'd)

```
/*--------------------------------------------------------------------*/
/*  Program chapter1_1                                                */
/*                                                                    */
/*  This program computes the                                         */
/*  distance between two points.                                      */

#include <stdio.h>
#include <math.h>

int main(void)
{
   /*  Declare and initialize variables.  */
   double x1=1, y1=5, x2=4, y2=7,
          side_1, side_2, distance;

   /*  Compute sides of a right triangle.  */
   side_1 = x2 - x1;
   side_2 = y2 - y1;
   distance = sqrt(side_1*side_1 + side_2*side_2);

   /*  Print distance.  */
   printf("The distance between the two points is "
          "%5.2f \n",distance);

   /*  Exit program.  */
   return 0;
}
/*--------------------------------------------------------------------*/
```

Example 2 (cont'd)

5. Testing

- After compiling your program, run it and see if it gives the correct result.
- Your program should print out

```
The distance between two points is 3.61
```

- If not, what will you do?

37

Modification to Example 2

How will you find the distance between two other points (2,5) and (10,8)?

```
/*------------------------------------------------------------------*/
/*  Program chapter1_1                                              */
/*                                                                  */
/*  This program computes the                                       */
/*  distance between two points.                                    */

#include <stdio.h>
#include <math.h>

int main(void)
{
   /*  Declare and initialize variables.  */
   double x1=1, y1=5, x2=4, y2=7,
          side_1, side_2, distance;

   /*  Compute sides of a right triangle.  */
   side_1 = x2 - x1;
   side_2 = y2 - y1;
   distance = sqrt(side_1*side_1 + side_2*side_2);

   /*  Print distance.  */
   printf("The distance between the two points is "
          "%5.2f \n",distance);

   /*  Exit program.   */
   return 0;
}
/*------------------------------------------------------------------*/
```

x1=2, y1=5, x2=10, y2=8.

38