C PROGRAMMING

Books

- The C Programming Language, by Dennis Ritchie and Brain Kernighan
- Balaguruswamy, Programming in ANCI C”, 2nd Edition
- Herbert Schildt. “The Complete Reference C ”, 4th Edition

Compilers

- Gcc compiler
- Use any editor based on gcc compilers

  - codeblocks

  - dev C++
- Or you can use any online compiler like

  - codechef

  -and others

Your First

C PROGRAM !!!!!

Simplest C Program

Hash

```
#include<stdio.h>
 int main()
{
printf(“ Programming in C is Interesting”);
return 0;
}
```

Preprocessor Directive

- # include: Is a preprocessor directive which directs to include the designated file which contains the declaration of library functions (pre defined).

- stdio.h (standard input output) : A header file which contains declaration for standard input and output functions of the program.Like printf, scanf

When to use preprocessor directive

- When one or more library functions are used, the corresponding header file where the information about these functions will be present are to be included.
- When you want to make use of the functions(user-defined) present in a different program in your new program the source program of previous function has to be included.

Defining main( )

- When a C program is executed, system first calls the main( ) function, thus a C program must always contain the function main( ) somewhere.

- A function definition has:

```
      heading
        {
             declarations ;
             statements;
        }
```

Basic Structure of a C program

```
preprocessor directive
int main(  )
{
   declarations;
   ________________________;
   ________body__________;
   ________________________;
return 0;
}
```

Concept of Comment

- Comments are inserted in program to maintain the clarity and for future references. They help in easy debugging.

- Comments are NOT compiled, they are just for the programmer to maintain the readability of the source code.

Comments are included as

```
/* ..................
                 .....................................*/
```

Check this out!

```
#include<stdio.h>
int main()
{
printf(" India won the final cricket match in Sri
    Lanka ");
return 0;
}
```

Notion of keywords

- Keywords are certain reserved words, that have standard predefined meanings in C.
- All keywords are in lowercase.
- Some keywords in C:

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

Identifiers and Variables

- Identifier : A name has to be devised for program elements such as variables or functions.
- Variable:
- Variables are memory regions you can use to hold data while your program is running.
- Thus variable has an unique address in memory region.
- For your convenience, you give them names (instead of having to know the address)
- Because different types of data are different sizes, the computer needs to know what type each variable is – so it can reserve the memory you need

Rules for Identifiers

- Contains only letters, digits and under score characters,

  example amount,   hit_count
- Must begin with either a letter of the alphabet or an underscore character.
- Can not be same as the keywords, example it can not be void, int.
- Uppercase letters are different than lowercase, example amount, Amount and AMOUNT all three are different identifiers.
- Maximum length can be 31 characters.
- Should not be the same as one already defined in the library, example it can not be printf/scanf.
- No special characters are permitted. e.g. blank space,period, semicolon, comma etc.

PRE DEFINED DATA TYPES

Data Types

- Every program specifies a set of operations to be done on some data in a particular sequence.

- However, the data can be of many types such as a number, real, character, string etc.

- C supports many data types out of which the basic types are:

  int, float , double and char.

Four Basic Data Types

- In C, there are 4 basic data types:
  1. char,
  2. int,
  3. Float and
  4. Double

Each one has its own properties.For instance,they all have different sizes.

The size of a variable can be pictured as the number of memory slots that are required to store it.

Integer types

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Float types

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte or 12 byte	3.4E-4932 to 1.1E+4932	19 decimal places

```
#include <stdio.h>
int main(void)
{
        int integerType;
        float floatType;
        double doubleType;
         char charType;
        long double a;
        // Sizeof operator is used to evaluate the size of a variable
         printf("Size of int: %ld bytes\n",sizeof(integerType));
        printf("Size of float: %ld bytes\n",sizeof(floatType));
        printf("Size of double: %ld bytes\n",sizeof(doubleType));
        printf("Size of char: %ld byte\n",sizeof(charType));
        printf("Size of short: %ld byte\n",sizeof(short));
        printf("Size of long: %ld byte\n",sizeof(long));
        printf("Size of long double : %ld byte\n",sizeof(a));
        return 0;
}
```

Format specifiers

- There are several format specifiers-The one you use should depend on the type of the variable you wish to print out.The common ones are as follows:

Format Specifier	Type
%d	int
%c	char
%f	float
%lf	double
%s	string

To display a number in scientific notation,use %e.

To display a percentage sign,use %%

printf( )

1. It is used to print message or result on the output screen.It is define in stdio.h header file.
2. Returns the number of characters it outputs on the screen.

   Example:

printf(“Enter the number whose multiplication table you want to generate”);

printf(“ Balance in your account is %d”, bal); /* where bal is int type*/

printf(“Balance =%d,Total tax is %f ” ,bal, tax); /* where tax is float type */

scanf( )

- scanf( ) : It is used to input from the user numerical values, characters or strings.It is defined in stdio.h
- The function returns the number of data items that have been entered successfully.

  Example:

```
int num1,num2,num3;
char var;
printf(“ enter the numbers “);
  scanf(“%d%d%d”, &num1,&num2,&num3);
  printf(“enter a character”);
  scanf(“%c”, &var);
```

scanf( )

- Don’t try to display a decimal number using the integer format specifier,%d, as this displays unexpected values.
- Similarly,don’t use %f for displaying integers.
- Mixing %d with char variables, or %c with int variables is all right

scanf( )

- This function returns the number of data items that have been entered successfully

```
#include<stdio.h>
int main()
{
int n;
printf("enter the value");
printf("%d", scanf("%d",&n));
return 0;
}
```

Similarly Write a program to find out what does printf( ) returns????

Char Data type

- A variable of type char can store a single character.
- All character have a numeric code associated with them, so in reality while storing a character variable its numeric value gets stored.
- The set of numeric code is called as “ASCII”, American Standard Code for Information Interchange.
- Char takes 1 byte

ASCII codes

0	nul	20	dc4	40	(	60	<	80	P	100	d	120	x
1	soh	21	nak	41	)	61	=	81	Q	101	e	121	y
2	stx	22	syn	42	*	62	>	82	R	102	f	122	z
3	etx	23	etb	43	+	63	?	83	S	103	g	123	{
4	eot	24	can	44	,	64	@	84	T	104	h	124	|
5	enq	25	em	45	-	65	A	85	U	105	i	125	}
6	ack	26	sub	46	.	66	B	86	V	106	j	126	~
7	bel	27	esc	47	/	67	C	87	W	107	k	127	del
8	bs	28	fs	48	0	68	D	88	X	108	l		
9	ht	29	gs	49	1	69	E	89	Y	109	m		
10	nl	30	rs	50	2	70	F	90	Z	110	n		
11	vt	31	us	51	3	71	G	91	[	111	o		
12	np	32	sp	52	4	72	H	92	\	112	p		
13	cr	33	!	53	5	73	I	93	]	113	q		
14	so	34	“	54	6	74	J	94	^	114	r		
15	si	35	#	55	7	75	K	95	_	115	s		
16	dle	36	$	56	8	76	L	96	`	116	t		
17	dc1	37	%	57	9	77	M	97	a	117	u		
18	dc2	38	&	58	:	78	N	98	b	118	v		
19	dc3	39	‘	59	;	79	O	99	c	119	w		

Declaration of Variable

- To Declare a variable means to reserve memory space for it.

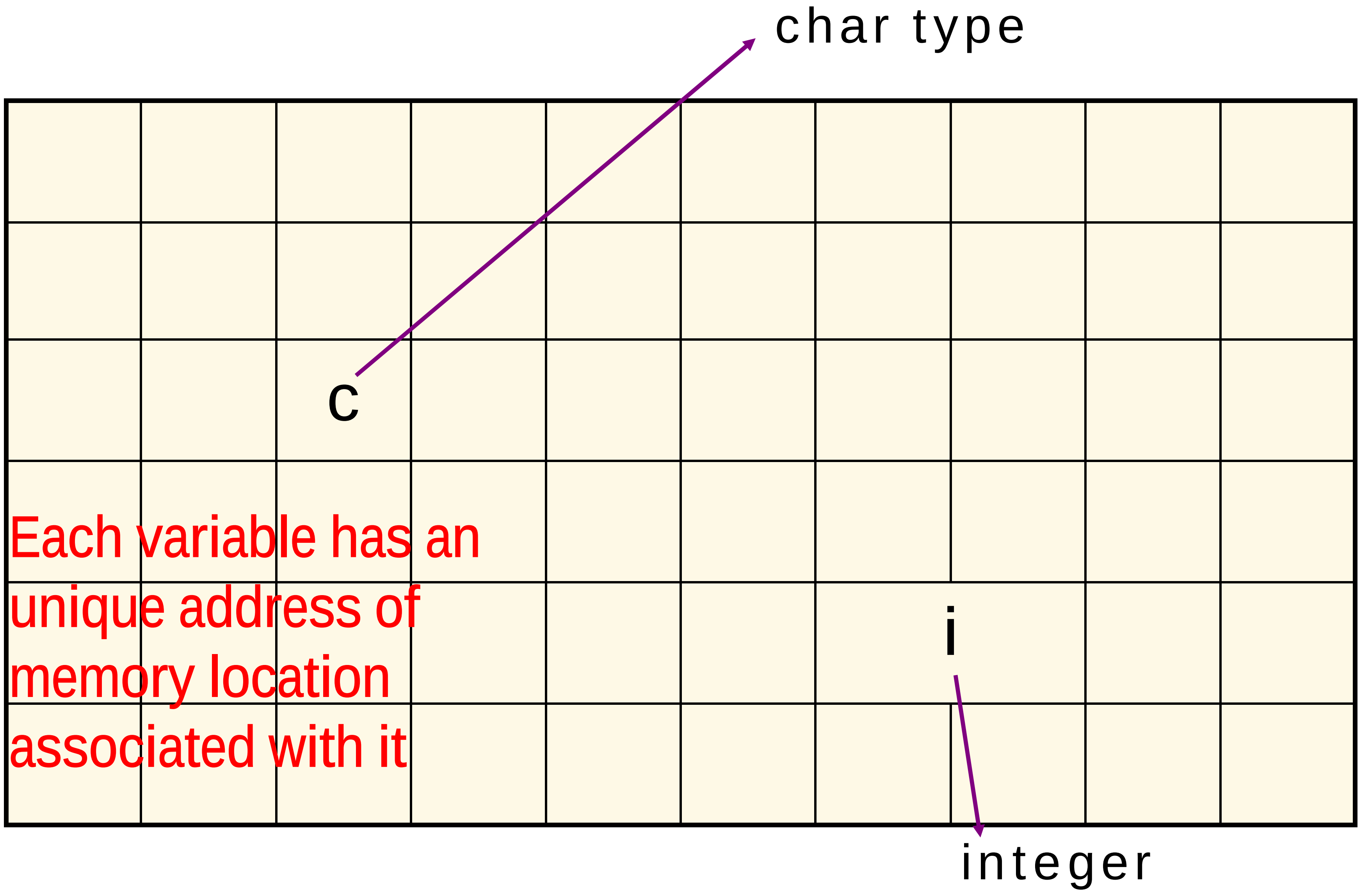
- In the declaration variable is provided a name which is used through out the program to refer to that character.

- Example:

```
char c;
OR
char c, k, l;
```

- Declaring the character does not initialize the variable with the desired value.

Memory view



Each variable has an unique address of memory location associated with it

Important about Variable

- Always remember in C , a variable must always be declared before it used.

- Declaration must specify the data type of the value of the variable.

- Name of the variable should match with its functionality /purpose.

  Example int count ; for counting the iterations.

  float per_centage ; for percentage of student

Char data type

- Characters (char) – To store a character into a char variable,you must enclose it with SINGLE QUOTE marks i.e. by ‘ ’.
- The double quotes are reserved for string(a collection of character).
- The character that you assign are called as character constants.
- You can assign a char variable an integer.i.e. its integer value.

  ‘a’, ‘z’ , ‘A’ , ‘Z’ , ‘?’, ‘@’, ‘0’, ‘9’

  - Special Characters: preceded by \

    ‘\n’, ‘\t’ , ‘\0’, ‘\’’, ‘\\’ *etc.*

Initialization

- Initializing a variable involves assigning(putting in) a value for the first time.This is done by using the ASSIGNMENT OPERATOR, denoted by the equals sign,=.

- Declaration and initializing can be done on separate lines, or on one line.

- char c=‘x’; or

  char c;

  c=‘x’

Printing value of char variable

```
printf( “%c”, a);
```

This causes the “ %c” to be replaced by value of character variable a.

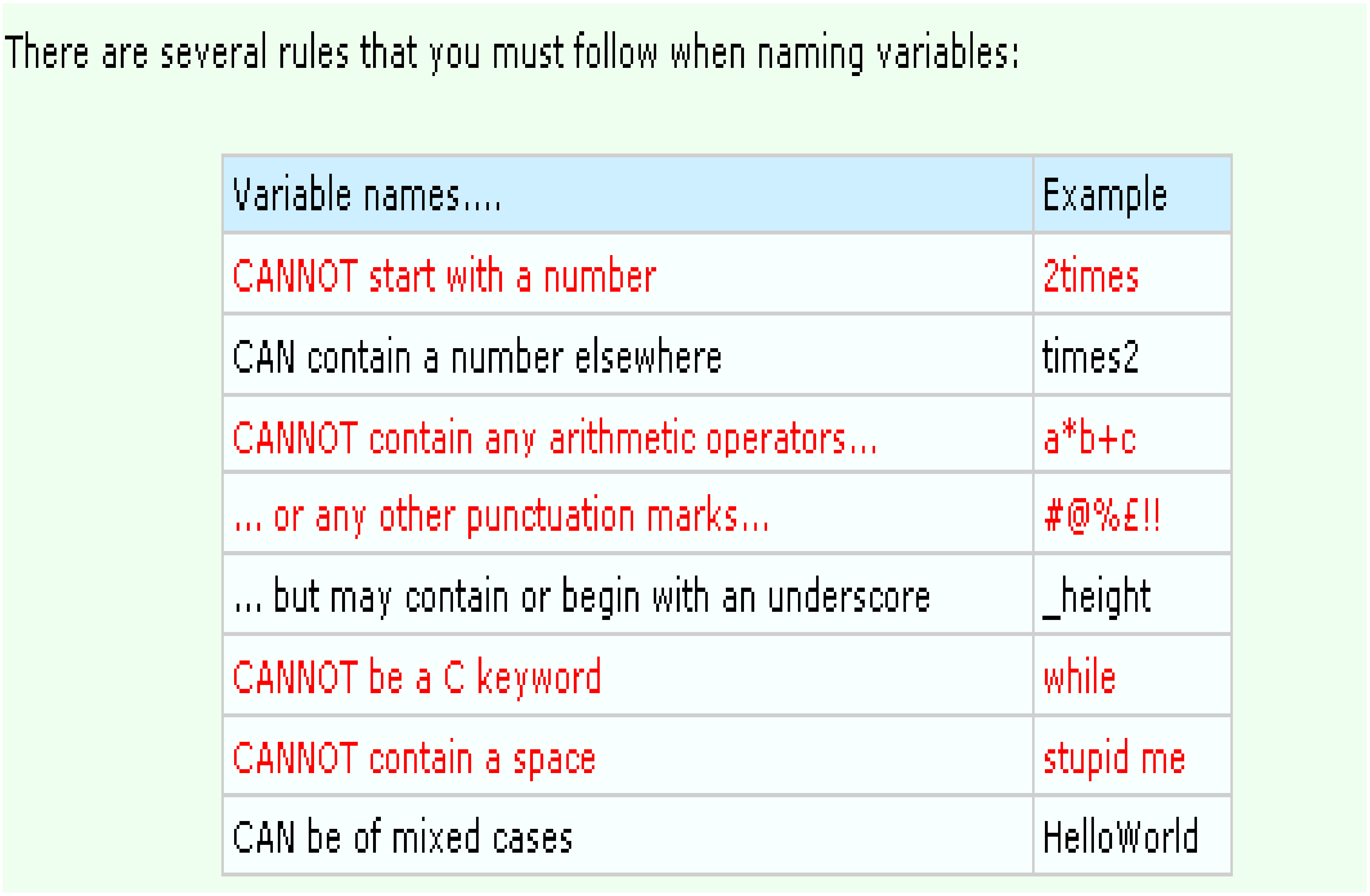
```
printf(“\n %c”, a);
```

“\n” is a new line character which will print the value of variable on newline.

Variables and Constants

- Variables are like containers in your computer’s memory-you can store values in them and retrieve or modify them when necessary
- Constants are like variables,but once a value is stored,its value can not be changed.

Naming Variables



There are several rules that you must follow when naming variables:

Variable names....	Example
CANNOT start with a number	2times
CAN contain a number elsewhere	times2
CANNOT contain any arithmetic operators...	a*b+c
... or any other punctuation marks...	#@%£!!
... but may contain or begin with an underscore	_height
CANNOT be a C keyword	while
CANNOT contain a space	stupid me
CAN be of mixed cases	HelloWorld

Expressions

- Expressions consist of a mixture of constants,variables and operators .They return values.
- Examples are:
  - 17

  -X*B/C+A
  - X+17

Program using character constants

```
#include<stdio.h>
int main()
    {
      char a,b,c,d;             /* declare char variables*/
      char e='o';               /* declare char variable*/
      a='H';                    /* initialize the rest */
      b='e';                    /* b=e is incorrect */
      c='l';                    /* c=“l” is incorrect*/
      d=108;                    /* the ASCII value of l is 108 */
      printf("%c%c%c%c%c",a,b,c,d,e);
      return 0;
      }
```

Integer Data Type

- Variables of the int data type represent whole numbers.
- If you try to assign a fraction to an int variable,the decimal part is ignored and the value assigned is rounded down from the actual value.
- Also assigning a character constant to an int variable assigns the ASCII value.

Program

```
#include<stdio.h>
main()
{     int a,b,c,d,e;
       a=10;
       b=4.3;
       c=4.8;
       d='A';
       e= 4.3 +4.8;
       printf("\n a=%d",a);
       printf("\n b=%d",b);
       printf("\n c=%d",c);
       printf("\n d=%d",d);
       printf("\n e=%d",e);
       printf("\n b+c=%d",b+c);
       return 0;
       }
```

```
a=10
b=4
c=4
 d=65
e=9
b+c=8
```

Float data type

- For storing decimal numbers float data type is used.
- Floats are relatively easy to use but problems tend to occur when performing division

In general:

An int divided by an int returns an int.

An int divided by a float returns a float.

A float divided by an int returns a float.

A float divided by a float returns a float.

int and float data types

- Integers (int) -- %d

  0   1   1000   -1   -10   666

- Floating point numbers (float) -- %f

  1.0   .1   1.0e-1   1e1

Program

```
#include<stdio.h>
main( )
    {
        return 0; float a=3.0;
        float b= 4.00 + 7.00;
        printf("a=%f ",a);
        printf("\n b=%f ",b);

    }
```

```
a=3.000000
b=11.000000
```

int and float

- Float is “ communicable type “
- Example:

```
1 + 2 * 3 - 4.0 / 5

= 1 + (2 * 3) - (4.0 / 5)

= 1 + 6 - 0.8

= 6.2
```

Example 2

```
(1 + 2) * (3 - 4) / 5

= ((1 + 2) * (3 - 4)) / 5

= (3 * -1) / 5

= -3 / 5

= 0
```

Multiple Format specifiers

- You could use as many format specifiers as you want with printf-just as long as you pass the correct number of arguments.
- The ordering of the arguments matters.The first one should corresponding to the first format specifier in the string and so on. Take this example:

  printf(“a=%d,b=%d, c=%d\n”, a , b, c);

  If a,b and c were integers,this statement will print the values in the correct order.
- Rewriting the statement as

  printf(“a=%d,b=%d, c=%d\n”, c,a,b);

Would still cause the program to compile OK,but the values of a,b and c would be displayed in the wrong order.

Statements

STATEMENTS are instructions and are terminated with a semicolon, `;`. Statements consist of a mixture of expressions, operators, function calls and various keywords. Here are some examples of statements:

```
x = 1 + 8;
printf("We will learn printf soon!\n");
int x, y, z; /* more on "int" later */
```

Statements

Which of these are valid:

int = 314.562 * 50;	Not valid
3.14 * r * r = area;	Not valid
k = a * b + c(2.5a + b);	Not valid
m_inst = rate of int * amt in rs;	Not valid
km = 4;	valid
area = 3.14 * r * r;	valid
S.I. = 400;	Not valid
sigma-3 = d;	Not valid

Statement Blocks

STATEMENT BLOCKS, on the other hand, can contain a group of statements. The C compiler compiles the statement block as if it was just one statement. To declare a statement block you enclose your statements between curly braces.

```
if(x==10)  {/* block 1 */
   printf("x equals 10\n");
   x = 11;
   printf("Now x equals 11\n");
   x = x + 1;
   printf("Now x equals 12\n");
} /* end of block 1 */
else { /* block 2 */
   printf("x not equal to 10\n");
   printf("Good bye!\n");
} /* end of block 2 */
```

Types of Constants

Numbers are considered as LITERAL constants - you can't change the number 20, nor can you assign something else into 20.

On the other hand, SYMBOLIC constants can be assigned a value during initialization and are represented by a word.

we'll use the `const` keyword.

```
const int radius = 5;
```

Since `radius` is declared using the `const` keyword, statements like: `radius = 12;` would be illegal.

OVERVIEW OF C

The C language

- General purpose and Structure programming language.
- Rich in library functions and allow user to create additional library functions which can be added to existing ones.
- Highly portable: Most C programs can run on many different machines with almost no alterations.
- It gives user the flexibility to create the functions, which give C immense power and scope.

C – Middle Level Language

- It combines both the powerful and best elements of high level languages as well as flexibility of assembly language.
- C resembles high level language such as PASCAL, FORTRAN as it contains keywords like if, else, for and while. It also uses control loops.
- C also possesses the low level features like pointers, memory allocation and bit manipulation

C – Structured Language

- C supports the breaking down of one big modules into smaller modules.

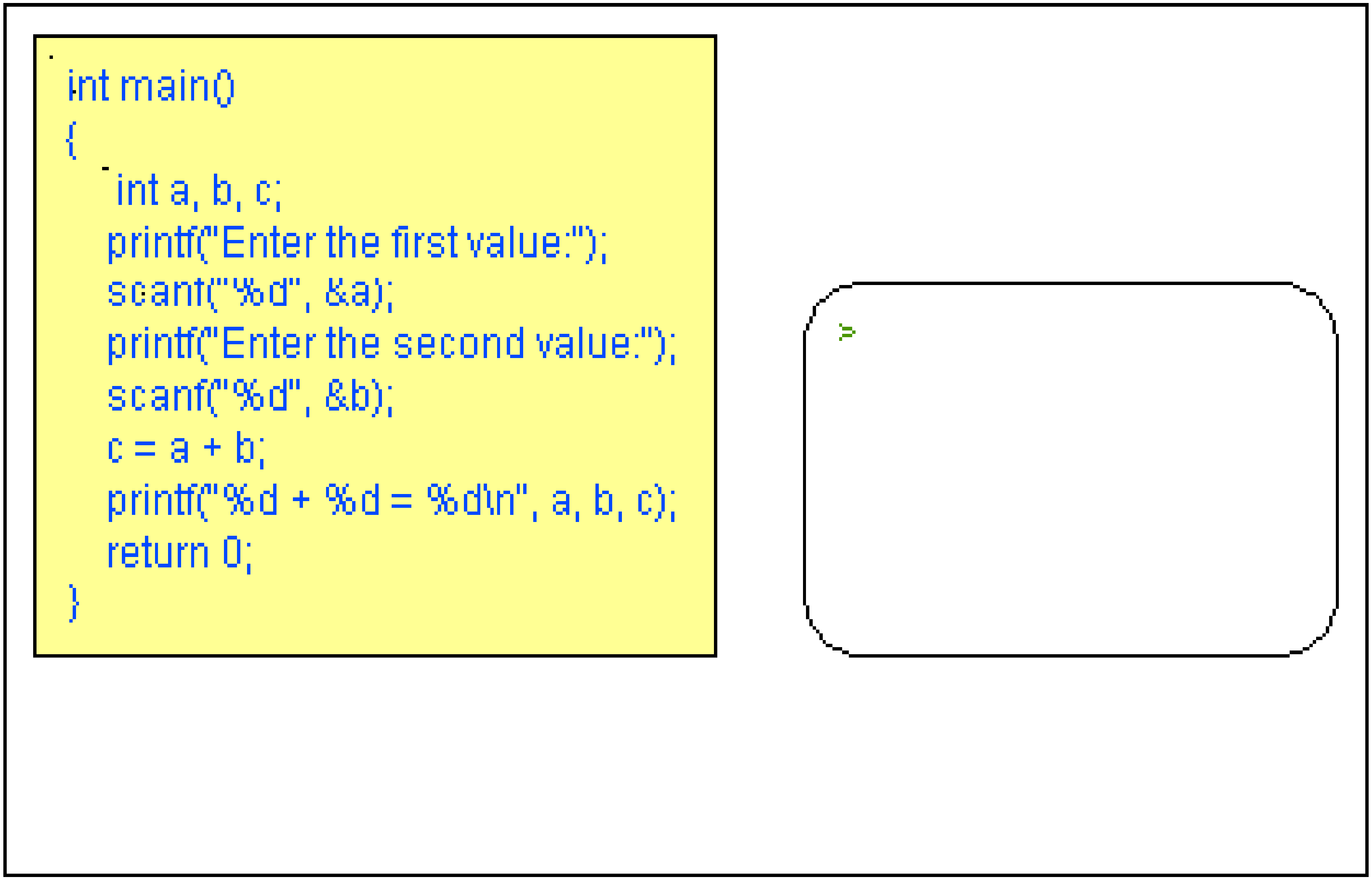
- It allows you to place statements anywhere on a line and does not require a strict field concept as in case of FORTARN or COBOL.

- It consist of functions which are building blocks of any program. They give the flexibility of separating tasks in a program and thus give modular programming approach.

Size of data types

- Size of data types is compiler and processor types dependent.
- The size **of data type** can be determined by using **sizeof** keyword **specified in between parenthesis**

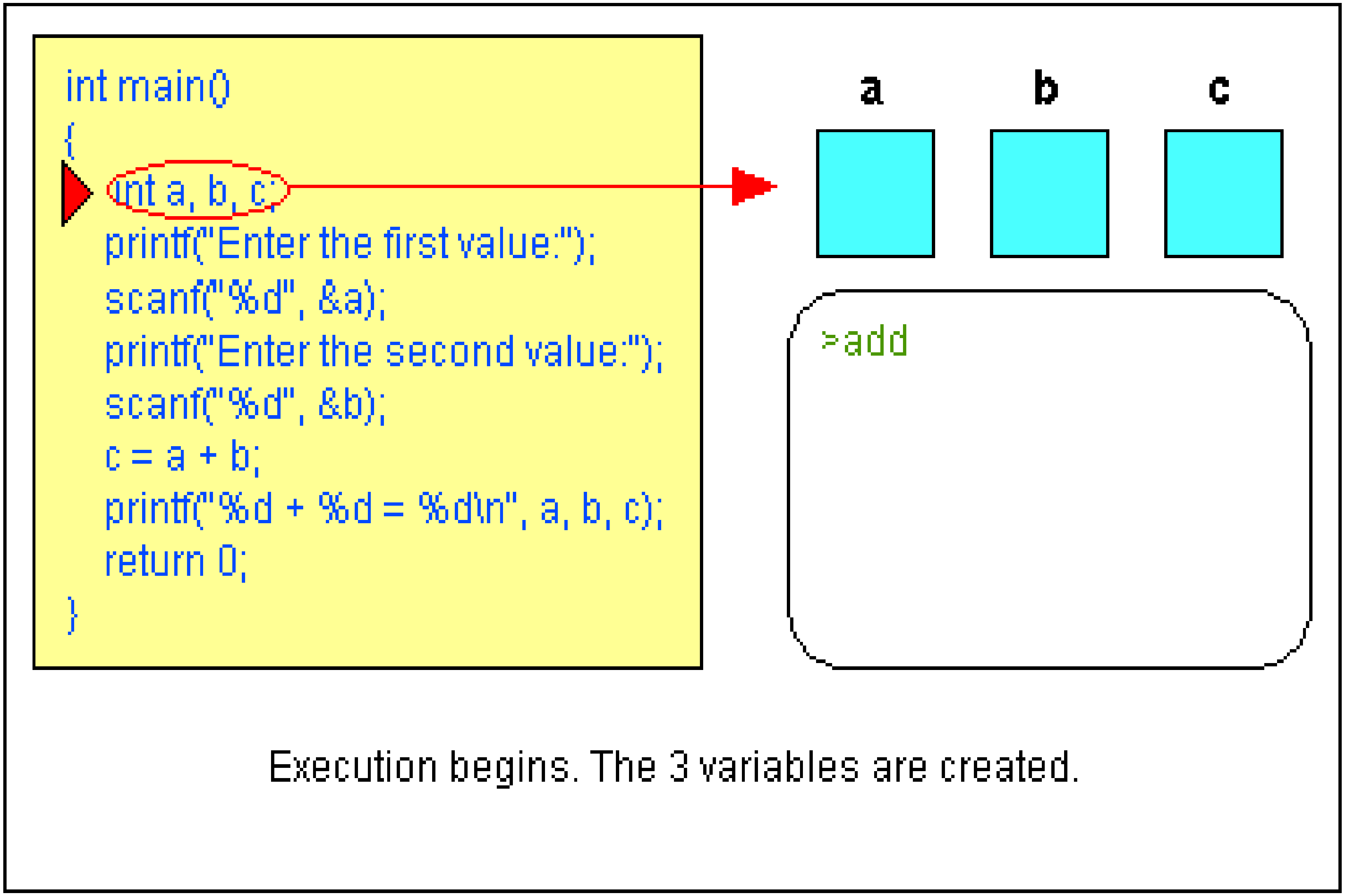
- For Turbo 3.0 compiler, usually in bytes
  - char -> 1 bytes
  - int -> 2 bytes
  - float -> 4 bytes
  - double -> 8 bytes



```c
int main()
{
    int a, b, c;
    printf("Enter the first value:");
    scanf("%d", &a);
    printf("Enter the second value:");
    scanf("%d", &b);
    c = a + b;
    printf("%d + %d = %d\n", a, b, c);
    return 0;
}
```

```
>
```

This animation shows the execution of a simple C program.

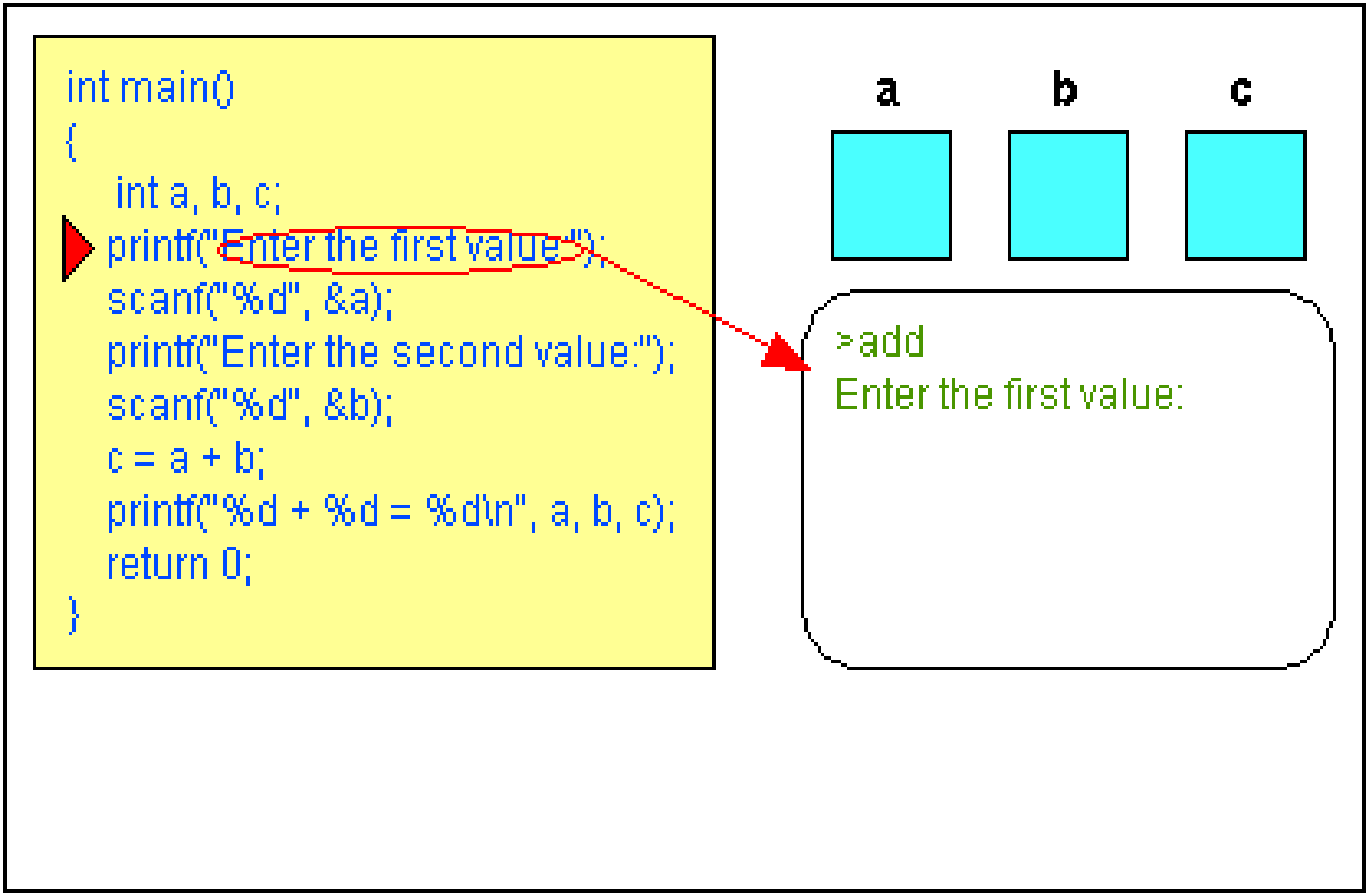


```
int main()
{
    int a, b, c;
    printf("Enter the first value:");
    scanf("%d", &a);
    printf("Enter the second value:");
    scanf("%d", &b);
    c = a + b;
    printf("%d + %d = %d\n", a, b, c);
    return 0;
}
```

```
>add
```

Execution begins. The 3 variables are created.

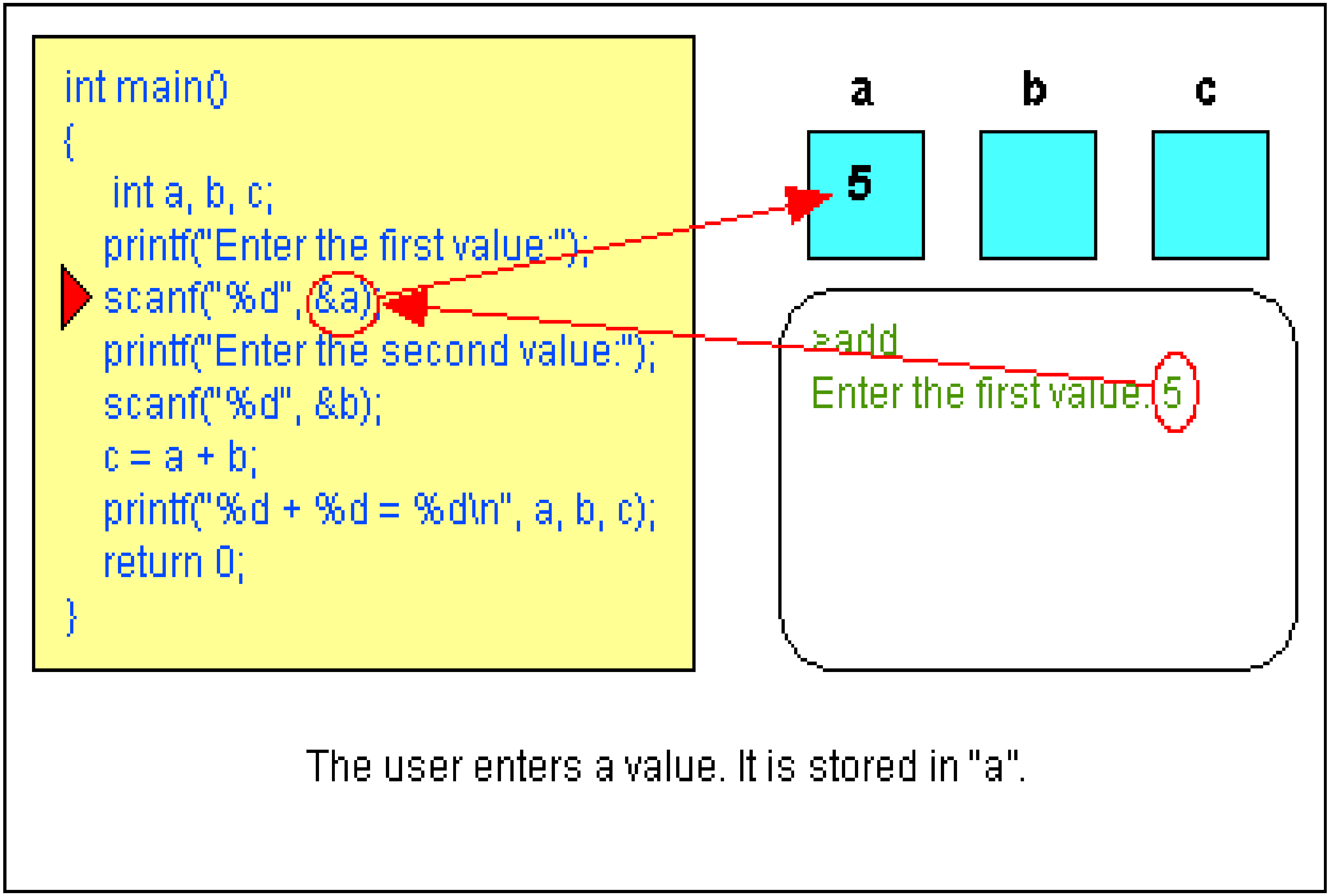
This animation shows the execution of a simple C program.



```
int main()
{
     int a, b, c;
    printf("Enter the first value:");
    scanf("%d", &a);
    printf("Enter the second value:");
    scanf("%d", &b);
    c = a + b;
    printf("%d + %d = %d\n", a, b, c);
    return 0;
}
```

```
>add
Enter the first value:
```

This animation shows the execution of a simple C program.

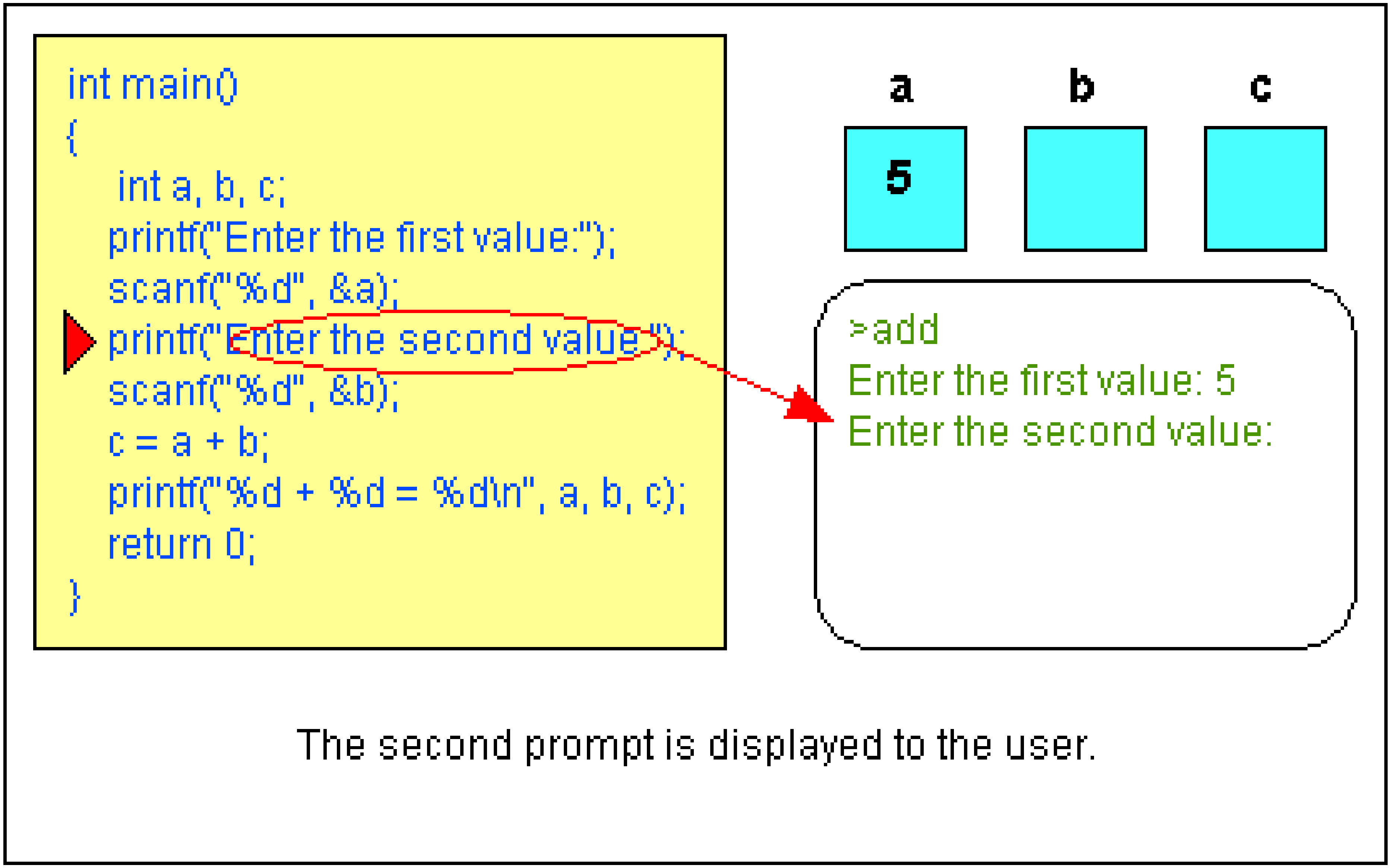


```
int main()
{
   int a, b, c;
  printf("Enter the first value:");
  scanf("%d", &a);
  printf("Enter the second value:");
  scanf("%d", &b);
  c = a + b;
  printf("%d + %d = %d\n", a, b, c);
  return 0;
}
```

```
>add
Enter the first value: 5
```

The user enters a value. It is stored in "a".

This animation shows the execution of a simple C program.

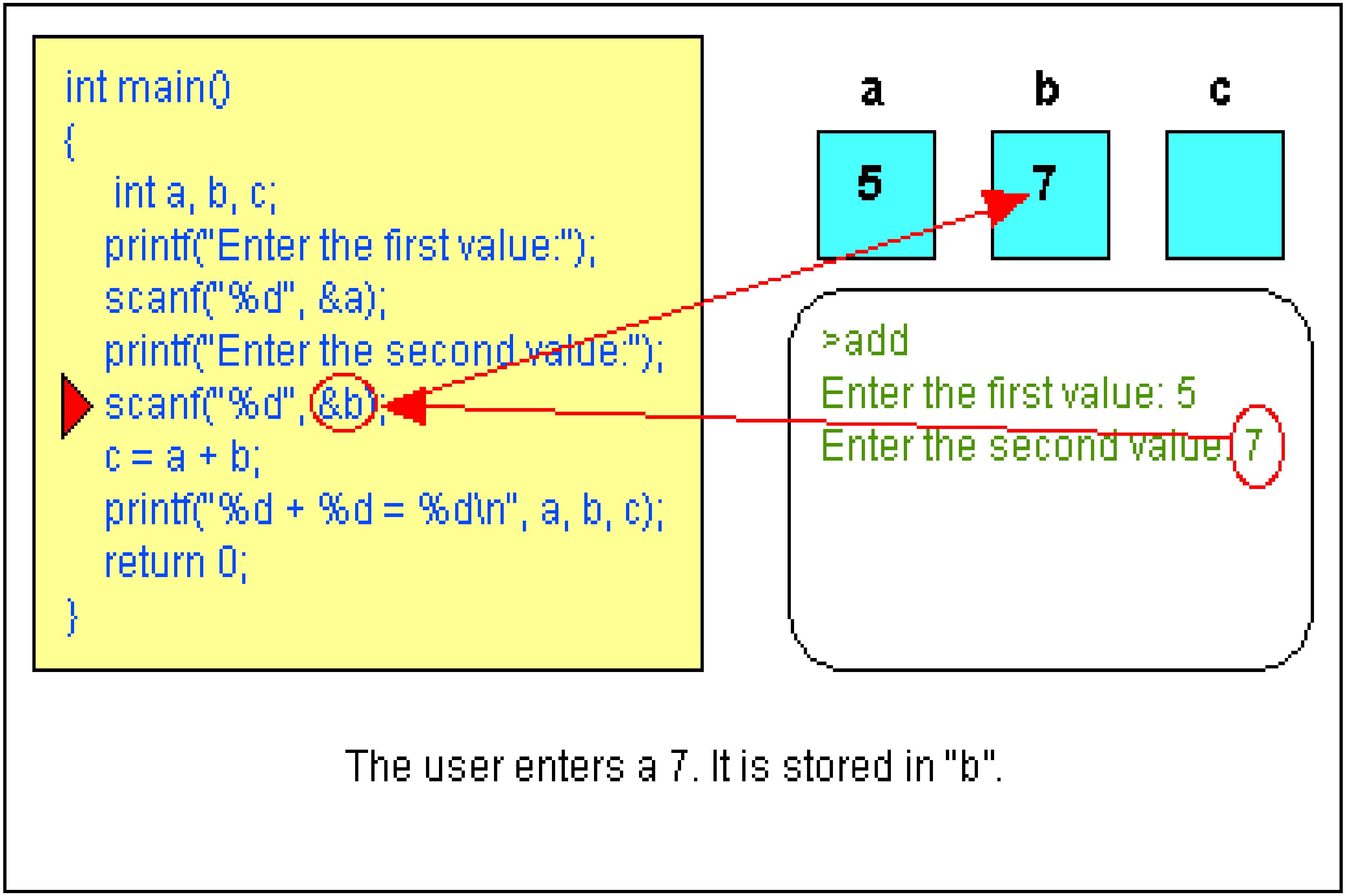


```
int main()
{
    int a, b, c;
  printf("Enter the first value:");
  scanf("%d", &a);
  printf("Enter the second value:");
  scanf("%d", &b);
  c = a + b;
  printf("%d + %d = %d\n", a, b, c);
  return 0;
}
```

```
>add
Enter the first value: 5
Enter the second value:
```

The second prompt is displayed to the user.

This animation shows the execution of a simple C program.

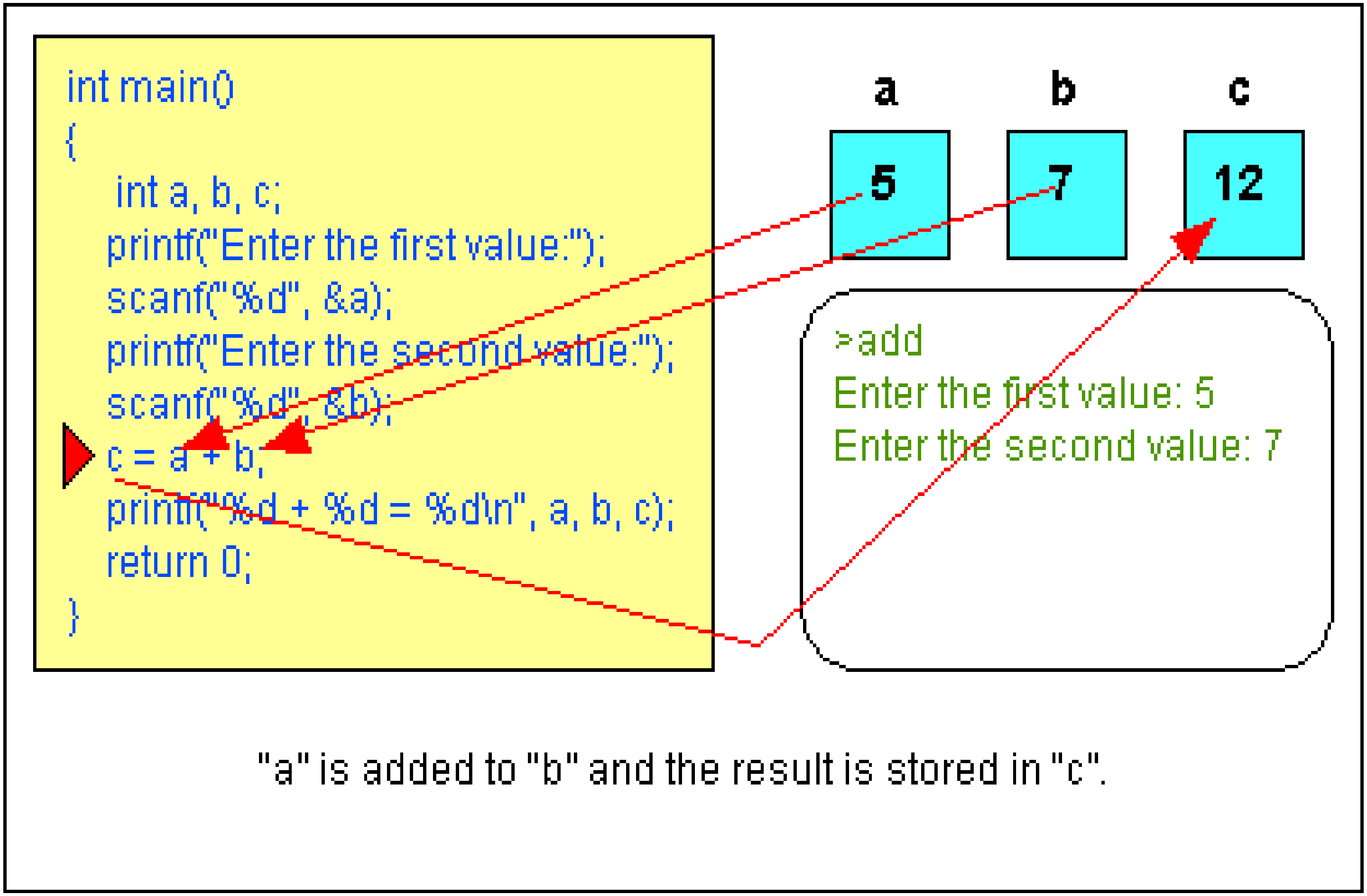


```
int main()
{
     int a, b, c;
    printf("Enter the first value:");
    scanf("%d", &a);
    printf("Enter the second value:");
    scanf("%d", &b);
    c = a + b;
    printf("%d + %d = %d\n", a, b, c);
    return 0;
}
```

```
>add
Enter the first value: 5
Enter the second value: 7
```

The user enters a 7. It is stored in "b".

This animation shows the execution of a simple C program.

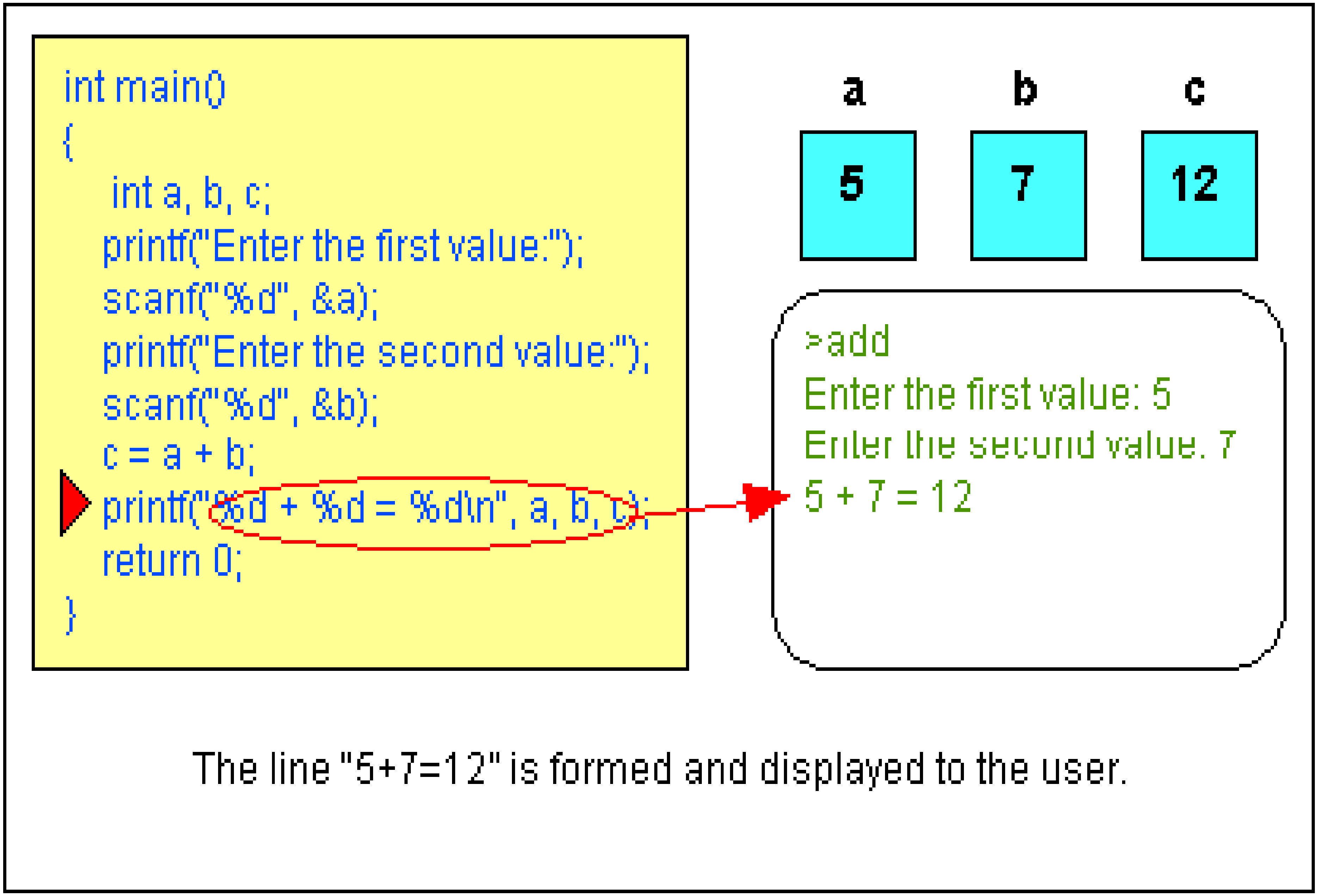


```
int main()
{
   int a, b, c;
  printf("Enter the first value:");
  scanf("%d", &a);
  printf("Enter the second value:");
  scanf("%d", &b);
  c = a + b;
  printf("%d + %d = %d\n", a, b, c);
  return 0;
}
```

```
>add
Enter the first value: 5
Enter the second value: 7
```

"a" is added to "b" and the result is stored in "c".

This animation shows the execution of a simple C program.

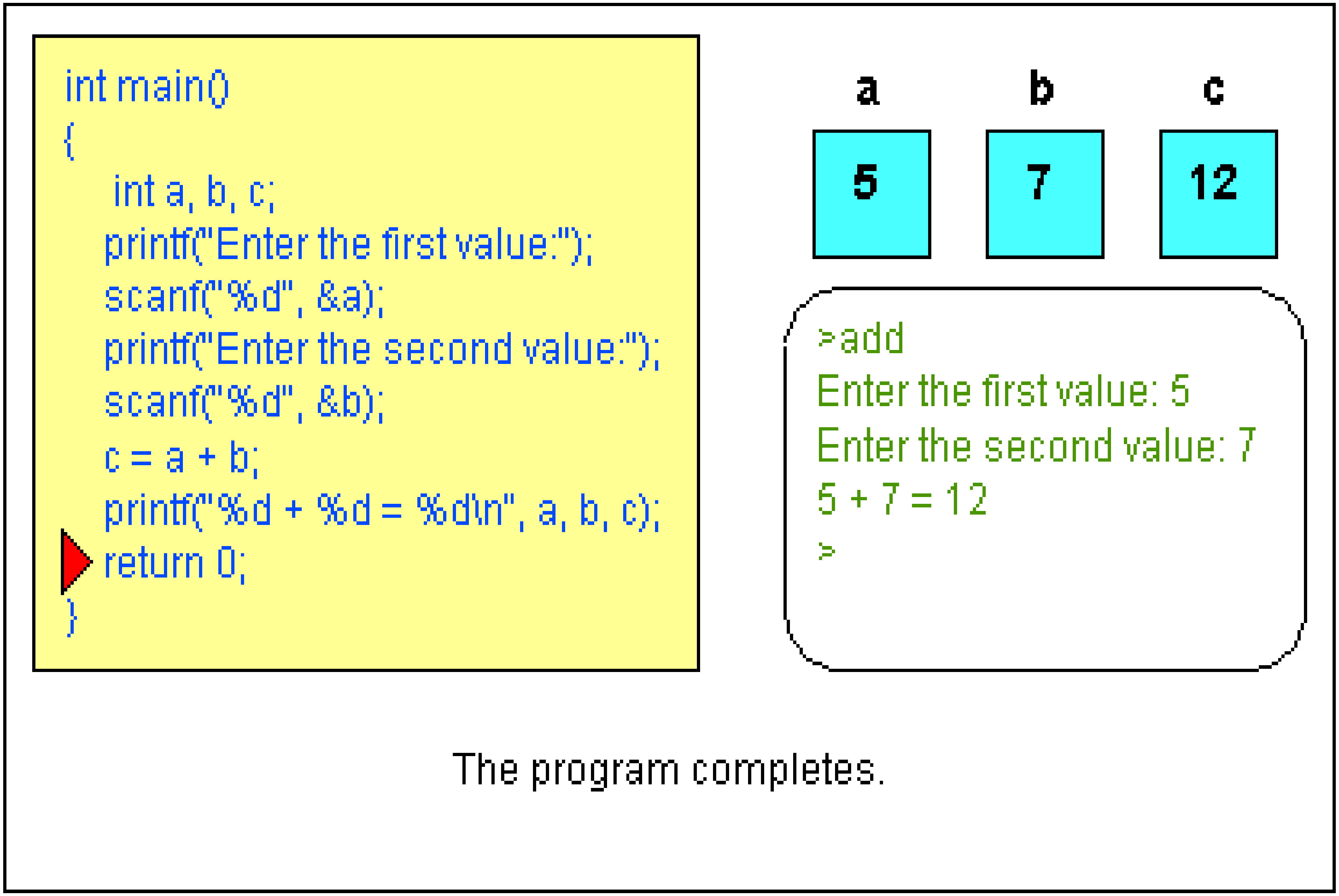


```
int main()
{
    int a, b, c;
   printf("Enter the first value:");
   scanf("%d", &a);
   printf("Enter the second value:");
   scanf("%d", &b);
   c = a + b;
   printf("%d + %d = %d\n", a, b, c);
   return 0;
}
```

```
>add
Enter the first value: 5
Enter the second value: 7
5 + 7 = 12
```

The line "5+7=12" is formed and displayed to the user.

This animation shows the execution of a simple C program.



```
int main()
{
    int a, b, c;
   printf("Enter the first value:");
   scanf("%d", &a);
   printf("Enter the second value:");
   scanf("%d", &b);
   c = a + b;
   printf("%d + %d = %d\n", a, b, c);
   return 0;
}
```

```
>add
Enter the first value: 5
Enter the second value: 7
5 + 7 = 12
>
```

The program completes.

This animation shows the execution of a simple C program.

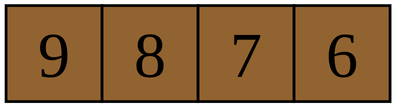
Output??

```
#include <stdio.h>
int main(void)
{ int a=72;
 char b='A';
 printf("a equals %d \n",a);
 printf("a equals %c \n",a);
 printf("b equals %d \n",b);
 printf("b equals %c \n",b);
return 0;
}
```

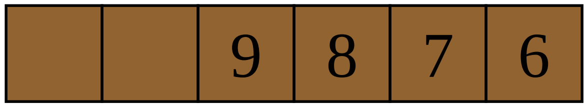
```
a equals 72
a equals H
b equals 65
b equals A
```

Example

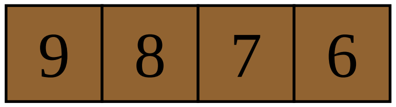
- printf(“%d”,9876)



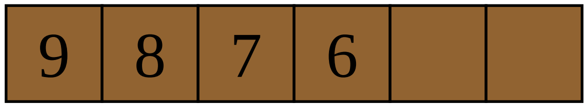
- printf(“%6d”,9876)



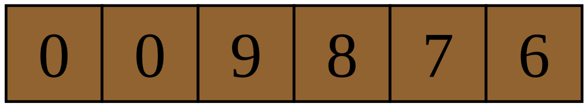
- printf(“%2d”,9876)



- printf(“%-6d”,9876)

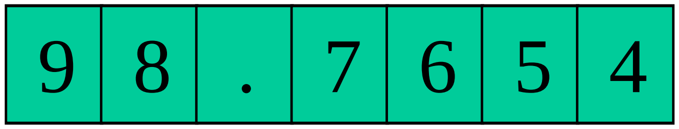


- printf(“%06d”,9876)

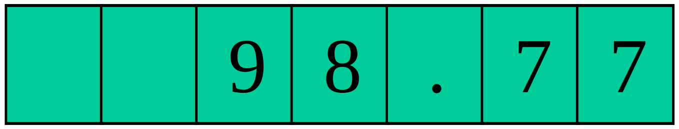


Example

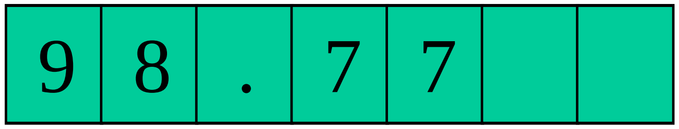
printf(“%7.4f”,98.7654)



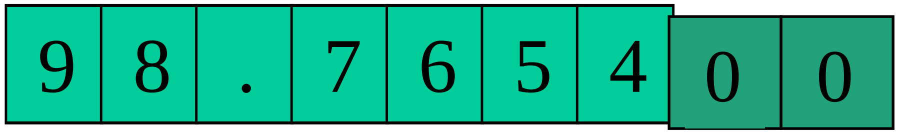
printf(“%7.2f”,98.7654)



printf(“%-7.2f”,98.7654)

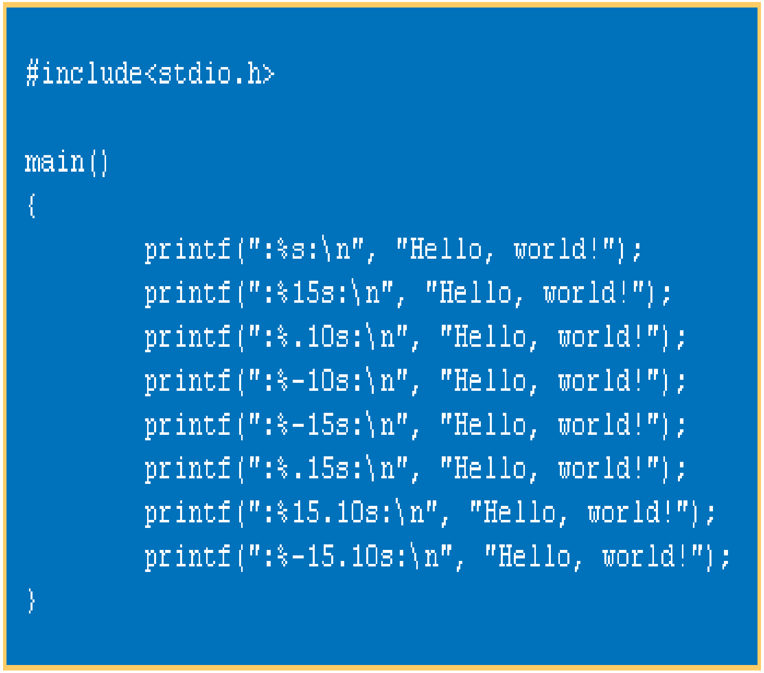


printf(“%f”,98.7654)



Note:-Using precision in a conversion specification in the format control string of a scanf statement is wrong.

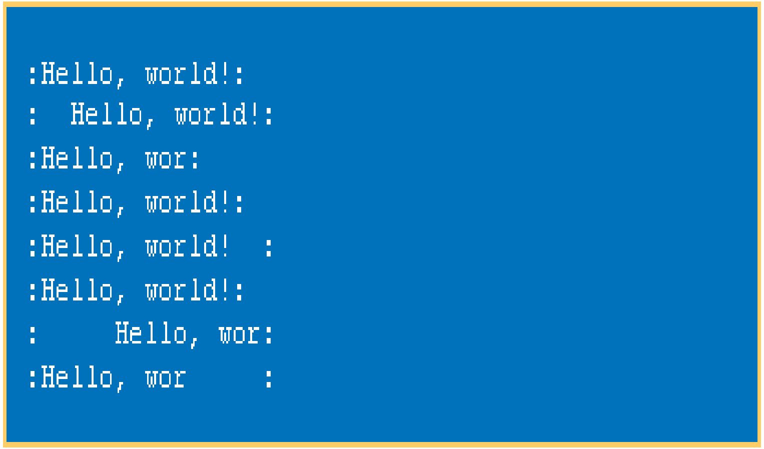
Formatting Strings



```
#include<stdio.h>

main()
{
        printf(":%s:\n", "Hello, world!");
        printf(":%15s:\n", "Hello, world!");
        printf(":%.10s:\n", "Hello, world!");
        printf(":%-10s:\n", "Hello, world!");
        printf(":%-15s:\n", "Hello, world!");
        printf(":%.15s:\n", "Hello, world!");
        printf(":%15.10s:\n", "Hello, world!");
        printf(":%-15.10s:\n", "Hello, world!");
}
```

The output of the example above:

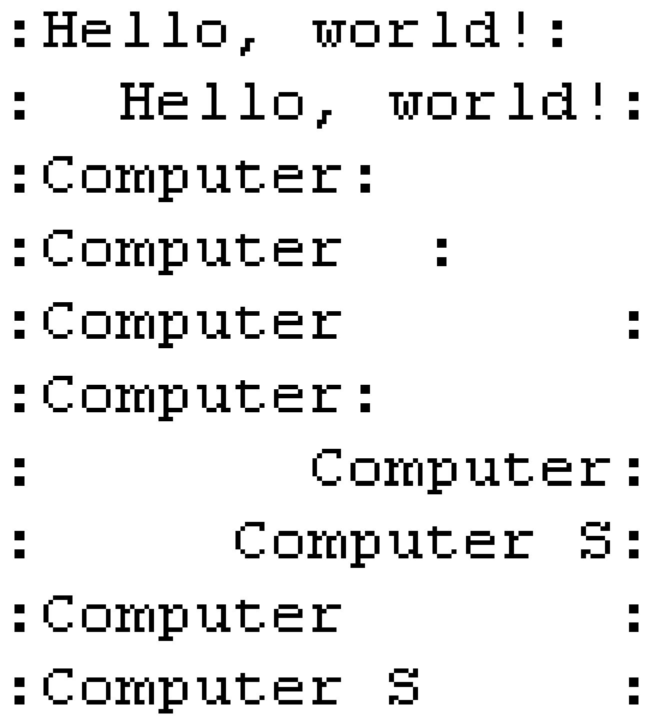


```
:Hello, world!:
:  Hello, world!:
:Hello, wor:
:Hello, world!:
:Hello, world!  :
:Hello, world!:
:     Hello, wor:
:Hello, wor     :
```

•The printf(":%s:\n", "Hello, world!"); statement prints the string (nothing special happens.)
•The printf(":%15s:\n", "Hello, world!"); statement prints the string, but print 15 characters. If the string is smaller the "empty" positions will be filled with "whitespace."
•The printf(":%.10s:\n", "Hello, world!"); statement prints the string, but print only 10 characters of the string.
•The printf(":%-10s:\n", "Hello, world!"); statement prints the string, but prints at least 10 characters. If the string is smaller "whitespace" is added at the end.
•The printf(":%-15s:\n", "Hello, world!"); statement prints the string, but prints at least 15 characters. The string in this case is shorter than the defined 15 character, thus "whitespace" is added at the end (defined by the minus sign.)
•The printf(":%.15s:\n", "Hello, world!"); statement prints the string, but print only 15 characters of the string. In this case the string is shorter than 15, thus the whole string is printed.
•The printf(":%15.10s:\n", "Hello, world!"); statement prints the string, but print 15 characters.
If the string is smaller the "empty" positions will be filled with "whitespace." But it will only print a maximum of 10 characters, thus only part of new string (old string plus the whitespace positions) is printed.
•The printf(":%-15.10s:\n", "Hello, world!"); statement prints the string, but it does the exact same thing as the previous statement, accept the "whitespace" is added at the end.

Example

```
#include<stdio.h>
main()
{
   printf(":%s:\n", "Hello, world!");
     printf(":%15s:\n", "Hello, world!");
     printf(":%.10s:\n", "Computer");
     printf(":%-10s:\n", "Computer");
     printf(":%-15s:\n", "Computer");
     printf(":%.15s:\n", "Computer");
     printf(":%15.10s:\n", "Computer");
   printf(":%15.10s:\n", "Computer Science");
     printf(":%-15.10s:\n", "Computer");
   printf(":%-15.10s:\n", "Computer Science");
}
```



```
:Hello, world!:
:  Hello, world!:
:Computer:
:Computer  :
:Computer       :
:Computer:
:       Computer:
:     Computer S:
:Computer       :
:Computer S     :
```

Backslash (\ ) character constants

- \n : To include new line
- \b : backspace
- \r : carriage return
- \t : horizontal tab
- \a : alert
- \” : double quote
- \’ : single quote
- \v : vertical tab
- \ \ : backslash

Exercise

- Write a C program to compute the percentage of a student in five subjects.

  Input the marks in five subjects from the user.

  Maximum marks in all the subjects is 100.