VTC Academy
Think Ahead, Beyond Boundaries

BASIC PROGRAMMING LANGUAGE

LESSON 2

Variables, Data Types and Console Input/Output

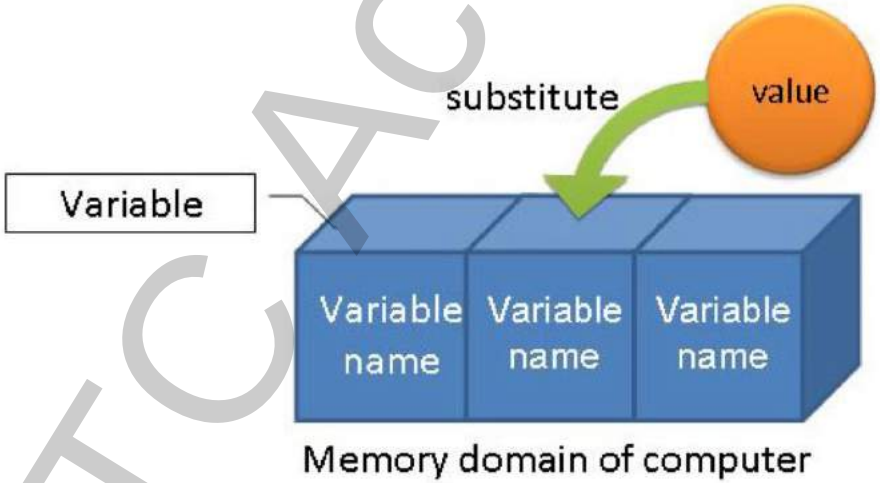
CONTENT

VTC Academy
Think Ahead, Beyond Boundaries

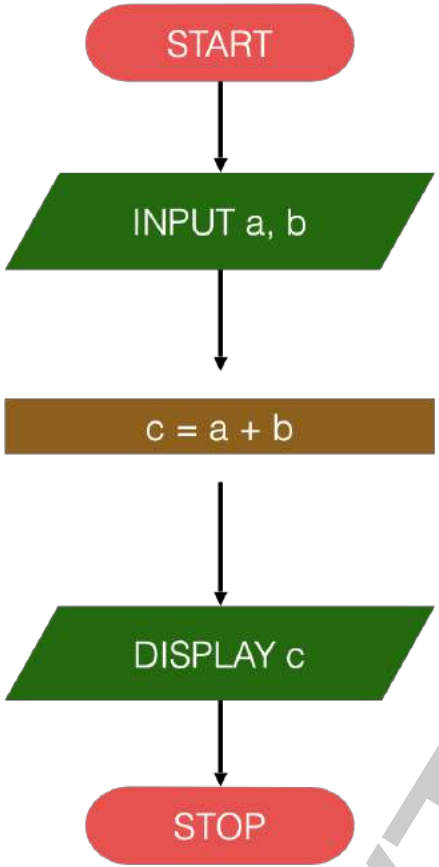
1. Introduction to Variable and Constant
2. Identifying Names
3. Different Between Variables and Constants
4. Data Types
5. Console Input / Ouput
   - Formated I/O: `scanf(), printf()`
   - Character I/O: `getchar(), putchar()`
6. Summary

Variable

- Variables allow to provide a meaningful name for the location in memory
- A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and the range of values that can be stored within that memory.



Variable Example



- a, b, c are variables in the flowchart
- Variable names takes away the need for a programmer to access memory locations using their address
- The operating system takes care of allocating space for the variables
- To refer to the value in the memory space, we need to only use the variable name

Constants

- A constant is a value whose worth never changes
- Examples:

5	numeric / integer constant
5.3	numeric / float constant
“Black”	string constant
‘C’	Character constant

- Variables hold constant values

Identifying Names

VTC Academy
Think Ahead, Beyond Boundaries

- The names of variables, functions, labels, and various other user defined objects are called identifiers
- Some correct identifier names: `Arena , s_count, marks40, class_one`
- Examples of erroneous identifiers: `1sttest, oh!god, start... end`
- Identifiers can be of any convenient length, but the number of characters in a variable that are recognized by a compiler varies from compiler to compiler
- Identifiers in C are case sensitive

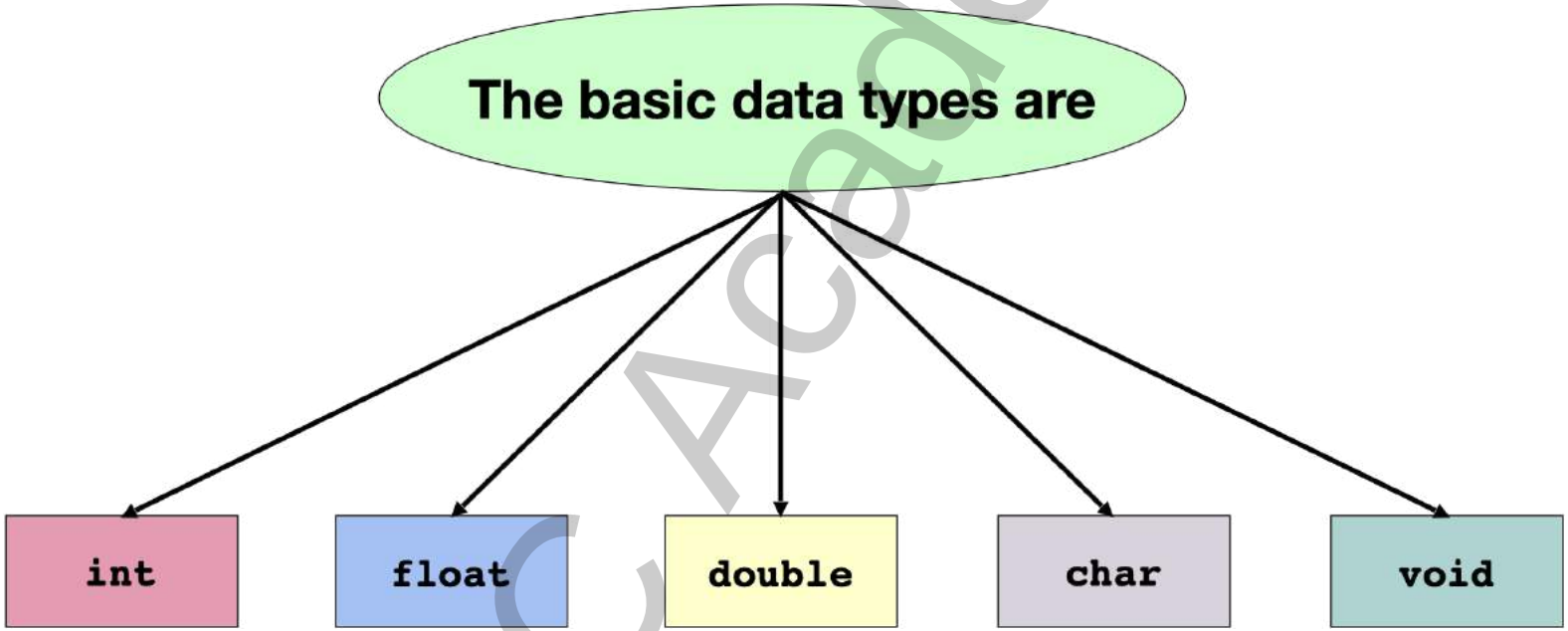
Data Types

- Data types in c refer to an extensive system used for declaring variables or functions of different types.
- The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.
- Different types of data are stored in variables.

Data Types

- Some examples are:
  - Numbers
    - Whole numbers, for example: 10 or 178993455
    - Real numbers, for example: 15.22 or 15463452.25
    - Positive numbers
    - Negative numbers
  - Text
    - For example: “John”
  - Logical values
    - For example: true or false

Data Types



Data Types

VTC Academy
Think Ahead, Beyond Boundaries

Data Types	Bytes	Range
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
signed char	1	-128 to 127
unsigned char	1	0 to 255
float	4	1.2E-38 to 3.4E+38
double	8	2.3E-308 to 1.7E+308

int Data Type

VTC Academy
Think Ahead, Beyond Boundaries

- Stores integer numeric data
- Cannot then store any other type of data like “Alan” or “12.7”
- Size of int Data Type depends on the Operating System
  - 16 bits (2 bytes) – Integers in the range -215 (-32768) to 215-1 (32767)
  - 32 bits (4 bytes) – Integers in the range -231 to 231-1
- Examples:
  - `int varName;`
  - `int i = 12322;`

float Data Type

VTC Academy
Think Ahead, Beyond Boundaries

- Stores values containing decimal places
- Precision of upto 6 digits
- 32 bits (4 bytes) of memory
- Examples:
  - `float f1;`
  - `float f2 = 23.05;`

double Data Type

VTC Academy
Think Ahead, Beyond Boundaries

- Stores values containing decimal places
- Precision of upto 17 digits
- 64 bits (8 bytes) of memory
- Examples:
  - `double d;`
  - `double pi = 3.141592653589;`

char Data Type

VTC Academy
Think Ahead, Beyond Boundaries

- Stores a single character of information
- 8 bits (1 byte) of memory
- Examples:
  - `char ch;`
  - `char gender='M';`

void Data Type

- Stores nothing
- Indicates the compiler that there is nothing to expect
- Use in case the function does not return value

Derived Data Types

VTC Academy
Think Ahead, Beyond Boundaries

- Data Type Modifiers + Basic Data Types = Derived Data Types
  - `unsigned + int = unsigned int // Permits only positive numbers`
  - `short + int = short int / short // Occupies less memory space than int`
  - `long + int = long int / long`
  - `long + double = long double // Occupies more space than int/double`

Standard Input / Output

- The standard library has functions for I/O that handle input, output, and character and string manipulation
- Standard input is usually the keyboard
- Standard output is usually the monitor (also called the console)
- Input and Output can be rerouted from or to files instead of the standard devices

The Header File <stdio.h>

VTC Academy
Think Ahead, Beyond Boundaries

- `stdio.h` is a file and is called the header file
- Contains the macros for many standard the input / output functions used in C program
- You must put following statement in top of program:
  - `#include <stdio.h>  // This is a preprocessor command`
- `printf(), scanf(), putchar(), getchar()` functions are designed and declared in `stdio.h` for proper execution

printf() Function

VTC Academy Think Ahead, Beyond Boundaries

- Used to display data on the standard output on the console screen
- Syntax:
  - `printf("control string", argument_list);`
- The argument list consists of constants, variables, expressions or functions separated by commas.
- The control string must always be enclosed within double quotes. It consists of constants, variables, expressions by commas.
  - Text characters: printable characters
  - Format Commands: % sign + format code
  - Nonprinting Characters: tabs, blanks and new lines

Format Code

VTC Academy
Think Ahead, Beyond Boundaries

Format	printf()	scanf()
Single Character	%c	%c
String	%s	%s
Signed decimal integer	%d	%d
Floating point (decimal notation)	%f	%f or %e
Floating point (decimal notation)	%lf	%lf
Floating point (exponential notation)	%e	%f or %e
Floating point (%f or %e , whichever is shorter)	%g	
Unsigned decimal integer	%u	%u
Unsigned hexadecimal integer (uses “ABCDEF”)	%x	%x
Unsigned octal integer	%o	%o

Control String Special Characters

VTC Academy Think Ahead, Beyond Boundaries

Display	String Format
To **display** \ character	`\\`
To **display** “ character	`\”`
To **display** % character	`%%`
To **display** a new line	`\n`
To **display** a tab	`\t`
To display data item will be left-justified	`-`
The default padding in a field is done with spaces. If the user wishes to pad a field with	`0`
To display integers as long int or a double	`%ld`
To display short integers	`h`
does not want to specify the field width in advance	`*`

printf() Function Example

VTC Academy
Think Ahead, Beyond Boundaries

```c
#include <stdio.h>

int main()
{
    printf("The number 555 in various forms:\n");
    printf("Without any modifier: \n");
    printf("[%d]\n",555);
    printf("With - modifier :\n");
    printf("[%-d]\n",555);
    printf("With digit string 10 as modifier :\n");
    printf("[%10d]\n",555);
    printf("With 0 as modifier : \n");
    printf("[%0d]\n",555);
    printf("With 0 and digit string 10 as modifiers :\n");
    printf("[%010d]\n",555);
    printf("With -, 0 and digit string 10 as modifiers: \n");
    printf("[%-010d]\n",555);
    printf("Print with floating point variable: \n");
    printf("[%010.2f]\n", 555.55555);

    return 0;
}
```

scanf() Function

- scanf() is used to accept data. The general format of scanf() function:
  - `scanf("control string", argument_list);`
- The format used in the printf() statement are used with the same syntax in the scanf() statements too.

	printf()	scanf()
Argument List	uses variable names, constants, symbolic constants and expression	uses pointers to variables
%g	Yes	No
%f and **%e**	Different	Same

scanf() Function Example

VTC Academy
Think Ahead, Beyond Boundaries

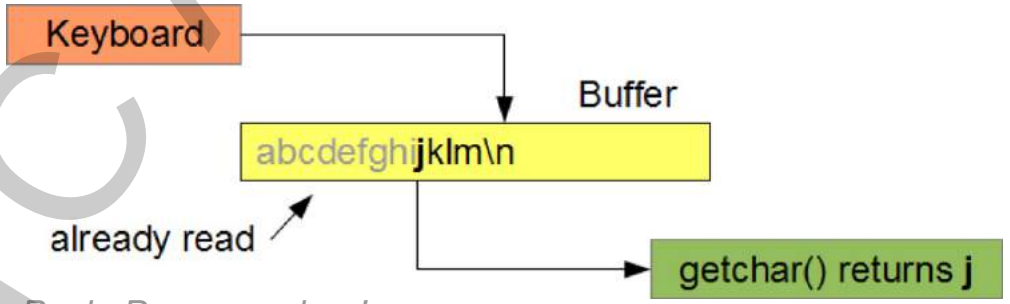
```c
#include <stdio.h>

int main()
{
    int i;
    printf("Input an integer number: ");
    scanf("%d", &i);
    printf("Integer number accepted: %d\n", i);

    return 0;
}
```

Buffered I/O

- A buffer is a temporary storage area, either in the memory, or on the controller card for the device
- Used to read and write ASCII characters
- Buffered I/O can be further subdivided into: Console I/O, Buffered File I/O
- Console I/O functions direct their operations to the standard input and output of the system In 'C' the simplest console I/O functions are:
  - `getchar()`
  - `putchar()`



getchar() Function

- The C library function `int getchar(void)` gets a character (an unsigned char) from stdin
- Has no argument, but the parentheses - must still be present
- Used to read input data, a character at a time from the keyboard
- Buffers characters until the user types a carriage return

getchar() Function Example

VTC Academy
Think Ahead, Beyond Boundaries

```c
#include <stdio.h>

int main()
{
    char letter;
    printf("\nPlease enter any character:");
    letter = getchar();
    printf("\nThe character entered by you is %c\n", letter);

    return 0;
}
```

putchar() Function

VTC Academy Think Ahead, Beyond Boundaries

- The C library function int putchar(int char) writes a character (an unsigned char) specified by the argument char to stdout.
- Syntax: `int putchar(int char)`

Argument	Function	Effect
character variable	`putchar(c)`	Displays the contents of character variable c
character constant	`putchar('A')`	Displays the letter A
numeric constant	`putchar('5')`	Displays the digit 5
escape sequence	`putchar('\t')`	Inserts a tab space character at the cursor position
escape sequence	`putchar('\n')`	Inserts a carriage return at the cursor position

putchar() Function Example

VTC Academy
Think Ahead, Beyond Boundaries

```c
#include <stdio.h>

int main()
{
    putchar('H');   putchar('\n');
    putchar('\t');
    putchar('E');   putchar('\n');
    putchar('\t');  putchar('\t');
    putchar('L');   putchar('\n');
    putchar('\t');  putchar('\t'); putchar('\t');
    putchar('L');   putchar('\n');
    putchar('\t');  putchar('\t'); putchar('\t');
    putchar('\t');
    putchar('O');   putchar('\n');

    return 0;
}
```

sizeof() Operator

VTC Academy
Think Ahead, Beyond Boundaries

- The sizeof operator is the most common operator in C.
- It is a compile-time unary operator and used to compute the size of its operand. It returns the size of a variable.
- It can be applied to any data type, float type, pointer type variables.
- When `sizeof()` is used with the data types, it simply returns the amount of memory allocated to that data type.
- The output can be different on different machines (32-bit and 64-bit system can show different of same data types).

sizeof() Operator Example

VTC Academy
Think Ahead, Beyond Boundaries

```c
#include <stdio.h>

int main()
{
    int size;

    size = sizeof(int);
    printf("\nint: %d bytes\n", size);
    size = sizeof(long int);
    printf("long int: %d bytes\n", size);
    size = sizeof(double);
    printf("double: %d bytes\n", size);
    size = sizeof(long double);
    printf("long double: %d bytes\n", size);

    return 0;
}
```

Summary

- Variables refers to the memory location where a particular value is to be stored
- A constant is a value whose worth never changes
- Identifiers can not store special character like “@”, “#”, space,…
- The main data types of C are character, integer, float, double float and void
- Unsigned, short and long are the three modifiers available in C
- printf(), scanf(), getchar(), putchar() are standard IO functions

VTC Academy
Think Ahead, Beyond Boundaries