Programming in C

Sample C program #1

```
#include <stdio.h>
main()
    {
        printf (“\n Our first look at a C program \n”);
    }
```

Sample C program #2

```
/* Compute the sum of two integers */

#include <stdio.h>
main()
    {
         int   a, b, c;
         a = 10;
         b = 20;
         c = a + b;
         printf (“\n The sum of %d and %d is %d\n”,
                  a,b,c);
    }
```

Sample C program #3

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
    {
        int   a, b, c;
        scanf ("%d %d %d", &a, &b, &c);
        if  ((a>b) && (a>c))    /* Composite condition check */
            printf ("\n Largest is %d", a);
        else
            if  (b>c)           /* Simple condition check */
                printf ("\n Largest is %d", b);
            else
                printf ("\n Largest is %d", c);
    }
```

Sample C program #4

```
#include <stdio.h>
#define    PI    3.1415926

/* Compute the area of a circle */
main()
    {
        float   radius, area;
        float   myfunc (float radius);

        scanf ("%f", &radius);
        area = myfunc (radius);
        printf ("\n Area is %f \n", area);
    }
```

```
float   myfunc (float r)
    {
        float   a;
        a = PI * r * r;
        return (a);      /* return result */
    }
```

Introduction to C

- C is a general-purpose, structured programming language.
  - Resembles other high-level structured programming languages, such as Pascal and Fortran-77.
  - Also contains additional features which allow it to be used at a lower level.
- C can be used for applications programming as well as for systems programming.
- There are only 32 keywords and its strength lies in its built-in functions.
- C is highly portable, since it relegated much computer-dependent features to its library functions.

History of C

- Originally developed in the 1970's by Dennis Ritchie at AT&T Bell Laboratories.
  - Outgrowth of two earlier languages BCPL and B.
- Popularity became widespread by the mid 1980's, with the availability of compilers for various platforms.
- Standardization has been carried out to make the various C implementations compatible.
  - American National Standards Institute (ANSI)
  - GNU

Structure of a C program

- **Every C program consists of one or more functions.**
  - **One of the functions must be called *main*.**
  - **The program will always begin by executing the *main* function.**
- **Each function must contain:**
  - **A function *heading*, which consists of the function *name*, followed by an optional list of *arguments* enclosed in parentheses.**
  - **A list of argument *declarations*.**
  - **A *compound statement*, which comprises the remainder of the function.**

Contd.

- Each compound statement is enclosed within a pair of braces: ‘{‘ and ‘}’
  - The braces may contain combinations of elementary statements and other compound statements.
- Comments may appear anywhere in a program, enclosed within delimiters ‘/*’ and ‘*/’.
  - Example:

```
a = b + c;    /* ADD TWO NUMBERS */
```

Example of a Function

```
/* Compute the sum of two integers */

#include <stdio.h>
main()
    {
        int   a, b, c;

        a = 10;
        b = 20;
        c = a + b;
        printf ("\n The sum of %d and %d is %d\n",
                    a,b,c);
    }
```

Desirable Programming Style

- **Clarity**
  - The program should be clearly written.
  - It should be easy to follow the program logic.
- **Meaningful variable names**
  - Make variable/constant names meaningful to enhance program clarity.
    - ‘area’ instead of ‘a’
    - ‘radius’ instead of ‘r’
- **Program documentation**
  - Insert comments in the program to make it easy to understand.
  - Never use too many comments.

Contd.

- **Program indentation**
  - **Use proper indentation.**
  - **Structure of the program should be immediately visible.**

Indentation Example #1 :: Good Style

```
#include <stdio.h>
#define    PI    3.1415926
/* Compute the area of a circle */

main()
   {
       float   radius, area;
       float   myfunc (float radius);

       scanf ("%f", &radius);
       area = myfunc (radius);
       printf ("\n Area is %f \n", area);
   }
```

```
float   myfunc (float r)
   {
       float   a;
       a = PI * r * r;
       return (a);      /* return result */
   }
```

Indentation Example #1 :: Bad Style

```
#include <stdio.h>
#define    PI    3.1415926
/* Compute the area of a circle */
main()
{
float   radius, area;
float   myfunc (float radius);
scanf ("%f", &radius);
area = myfunc (radius);
printf ("\n Area is %f \n", area);
}
```

```
float   myfunc (float r)
{
float   a;
a = PI * r * r;
return (a);      /* return result */
}
```

Indentation Example #2 :: Good Style

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
   {
      int   a, b, c;
      scanf (“%d %d %d”, &a, &b, &c);
      if  ((a>b) && (a>c))                    /* Composite condition check */
         printf (“\n Largest is %d”, a);
      else
         if  (b>c)                            /* Simple condition check */
            printf (“\n Largest is %d”, b);
         else
            printf (“\n Largest is %d”, c);
   }
```

Indentation Example #2 :: Bad Style

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
int   a, b, c;
scanf (“%d %d %d”, &a, &b, &c);
if  ((a>b) && (a>c))   /* Composite condition check */
printf (“\n Largest is %d”, a);
else
if  (b>c)     /* Simple condition check */
printf (“\n Largest is %d”, b);
else
printf (“\n Largest is %d”, c);
}
```

The C Character Set

- **The C language alphabet:**
  - **Uppercase letters ‘A’ to ‘Z’**
  - **Lowercase letters ‘a’ to ‘z’**
  - **Digits ‘0’ to ‘9’**
  - **Certain special characters:**

!	#	%	^	&	*	(	)
-	_	+	=	~	[	]	\
|	;	:	‘	“	{	}	,
.	<	>	/	?	blank		

Identifiers and Keywords

- **Identifiers**
  - Names given to various program elements (variables, constants, functions, etc.)
  - May consist of *letters*, *digits* and the *underscore* ('_') character, with no space between.
  - First character must be a letter.
  - An identifier can be arbitrary long.
    - Some C compilers recognize only the first few characters of the name (16 or 31).
  - Case sensitive
    - 'area', 'AREA' and 'Area' are all different.

Contd.

- **Keywords**
  - Reserved words that have standard, predefined meanings in C.
  - Cannot be used as identifiers.
  - OK within comments.
  - Standard C keywords:

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

Valid and Invalid Identifiers

- **Valid identifiers**

```
X
abc
simple_interest
a123
LIST
stud_name
Empl_1
Empl_2
avg_empl_salary
```

- **Invalid identifiers**

```
10abc
my-name
“hello”
simple interest
(area)
%rate
```

Data Types in C

int :: integer quantity

Typically occupies 4 bytes (32 bits) in memory.

char :: single character

Typically occupies 1 bye (8 bits) in memory.

float :: floating-point number (a number with a decimal point)

Typically occupies 4 bytes (32 bits) in memory.

double :: double-precision floating-point number

Typically occupies 8 bytes (64 bits) in memory.

Contd.

- Some of the basic data types can be augmented by using certain data type qualifiers:
  - short
  - long
  - signed
  - unsigned
- Typical examples:
  - short int
  - long int
  - unsigned int

Some Examples of Data Types

- **int**

  0, 25, -156, 12345, ? 99820

- **char**

  ‘a’, ‘A’, ‘*’, ‘/’, ‘ ’

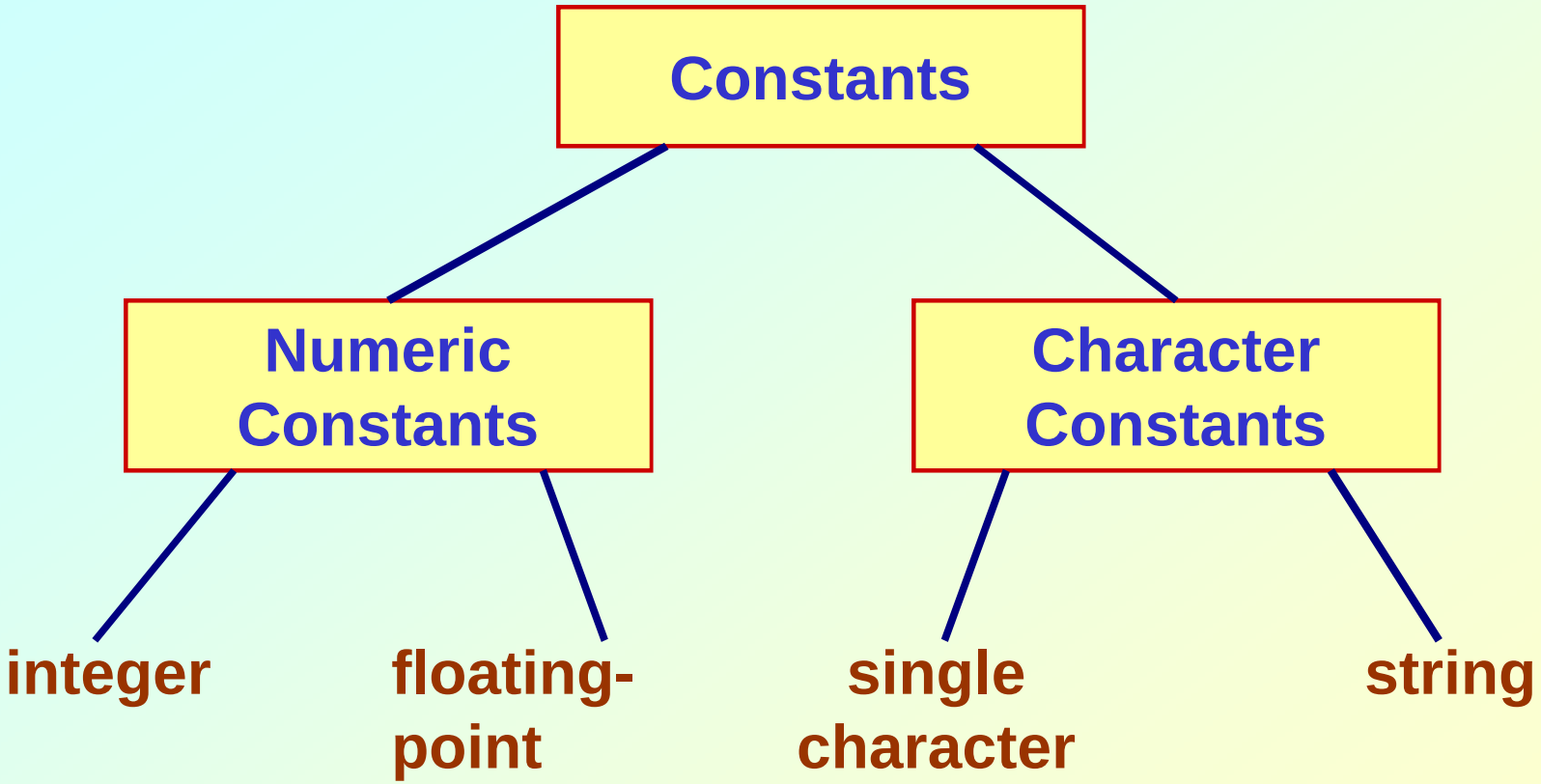
- **float**

  23.54, ? 0.00345, 25.0

  2.5E12, 1.234e-5

E or e means “10 to the power of”

Constants


Constants
Numeric Constants
Character Constants
integer
floating-point
single character
string

Integer Constants

- Consists of a sequence of digits, with possibly a plus or a minus sign before it.
  - Embedded spaces, commas and non-digit characters are not permitted between digits.
- Maximum and minimum values (for 32-bit representations)

  Maximum :: 2147483647 ($2^{31} - 1$)

  Minimum :: – 2147483648 (-2^{31})

Floating-point Constants

- Can contain fractional parts.
- Very large or very small numbers can be represented.

  23000000 can be represented as 2.3e7

- Two different notations:

  1. Decimal notation

     25.0, 0.0034, .84, -2.234

  2. Exponential (scientific) notation

     3.45e23, 0.123e-12, 123E2

e means "10 to the power of"

Single Character Constants

- Contains a single character enclosed within a pair of single quote marks.
  - Examples :: ‘2’, ‘+’, ‘Z’
- Some special backslash characters

‘\n’	new line
‘\t’	horizontal tab
‘\’’	single quote
‘\”’	double quote
‘\\’	backslash
‘\0’	null

String Constants

- **Sequence of characters enclosed in double quotes.**
  - **The characters may be letters, numbers, special characters and blank spaces.**
- **Examples:**

  “nice”, “Good Morning”, “3+6”, “3”, “C”
- **Differences from character constants:**
  - **‘C’ and “C” are not equivalent.**
  - **‘C’ has an equivalent integer value while “C” does not.**

Variables

- It is a data name that can be used to store a data value.
- Unlike constants, a variable may take different values in memory during execution.
- Variable names follow the same naming convention for identifiers.
  - Examples :: temp, speed, name2, current, my_salary

Example

```
int    a, b, c;
char   x;

a = 3;
b = 50;
c = a – b;
x = ‘d’;

b = 20;
a = a + 1;
x = ‘G’;
```

Declaration of Variables

- **There are two purposes:**
  1. **It tells the compiler what the variable name is.**
  2. **It specifies what type of data the variable will hold.**
- **General syntax:**

  data-type variable-list;
- **Examples:**

```
int   velocity, distance;
int   a, b, c, d;
float  temp;
char  flag, option;
```

A First Look at Pointers

- **A variable is assigned a specific memory location.**
  - **For example, a variable *speed* is assigned memory location *1350*.**
  - **Also assume that the memory location contains the data value *100*.**
  - **When we use the name *speed* in an expression, it refers to the value *100* stored in the memory location.**

```
distance = speed * time;
```

- **Thus every variable has an *address* in memory, and its *contents*.**

Contd.

- In C terminology, in an expression

  speed refers to the contents of the memory location.

  &speed refers to the address of the memory location.

- Examples:

```
printf (“%f %f %f”, speed, time, distance);
scanf (“%f %f”, &speed, &time);
```

An Example

```
#include <stdio.h>
main()
    {
        float  speed, time, distance;

        scanf (“%f %f”, &speed, &time);
        distance = speed * time;
        printf (“\n The distance traversed is: \n”,
                                              distance);
    }
```