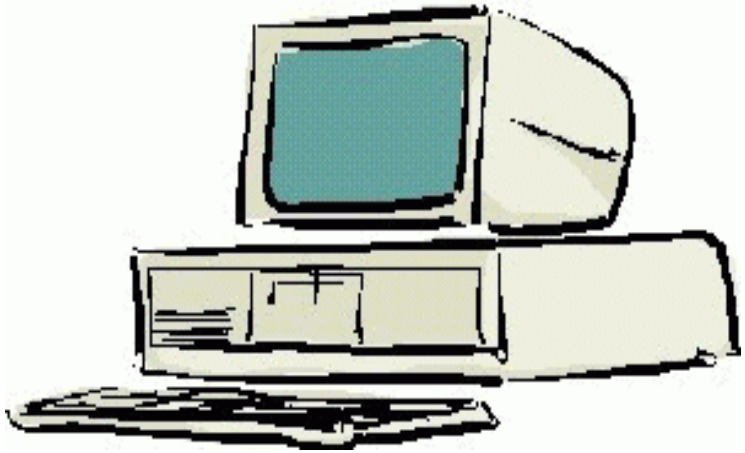
Programming in C



Session 1

Seema Sirpal
Delhi University Computer Centre

What is Programming

Computer Programming is the art of making a computer do what you want it to do.

At the very simplest level it consists of issuing a sequence of commands to a computer to achieve an objective.

What is Programming

A computer program is simply a set of instructions to tell a computer how to perform a particular task.

Like a recipe: a set of instructions to tell a cook how to make a particular dish.

It describes the ingredients (the data) and the sequence of steps (the process) needed to convert the ingredients into the cake or whatever.

Programs are very similar in concept.

What is Programming

Just as you speak to a friend in a language so you 'speak' to the computer in a language.

The only language that the computer understands is called *binary.*

Binary is unfortunately very difficult for humans to read or write so we have to use an intermediate language and get it translated into binary for us.

This is rather like watching the American and Russian presidents talking at a summit meeting - One speaks in English, then an interpreter repeats what has been said in Russian.

The thing that translates our intermediate language into binary is also called an interpreter

What is Programming

The very first programmers actually had to enter the binary codes themselves, this is known as *machine code* programming and is incredibly difficult.

The next stage was to create a translator that simply converted English equivalents of the binary codes into binary so that instead of having to remember that the code 001273 05 04 meant add 5 to 4 programmers could now write ADD 5 4.

This very simple improvement made life much simpler and these systems of codes were really the first programming languages, one for each type of computer.

They were known as *assembler* languages and *Assembler programming* is still used for a few specialized programming tasks today.

What is Programming

Even this was very primitive, it was still very difficult and took a lot of programming effort to achieve even simple tasks.

Gradually computer scientists developed higher level computer languages to make the job easier.

This was just as well because at the same time users were inventing ever more complex jobs for computers to solve!

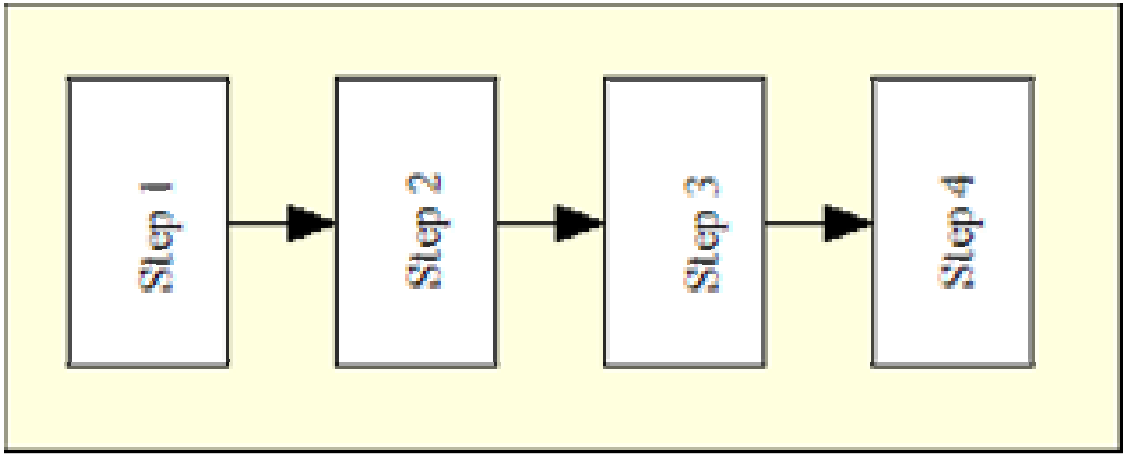
This competition between the computer scientists and the users is still going on and new languages keep on appearing.

This makes programming interesting but also makes it important that as a programmer you understand the concepts of programming as well as the pragmatics of doing it in one particular language.

Structured Programming

Sequences of instructions:

Here the program flows from one step to the next in strict sequence.

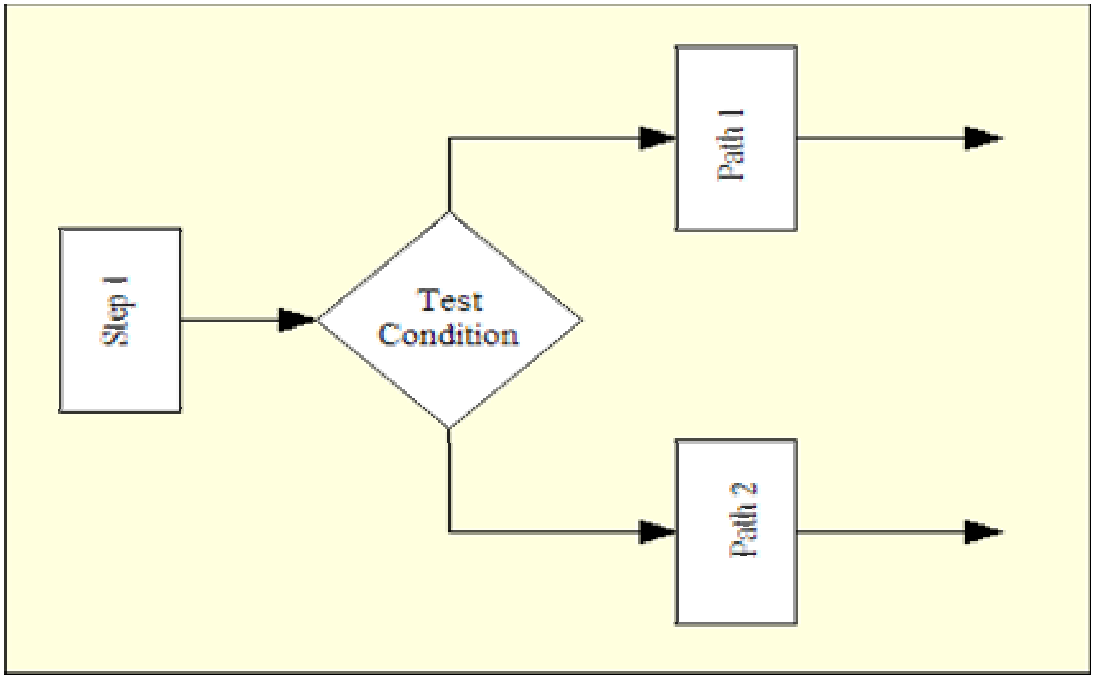


Structured Programming

Branches:

Here the program reaches a decision point and if the result of the test is true then the program performs the instructions in *Path 1*, and if false it performs the actions in *Path 2*.

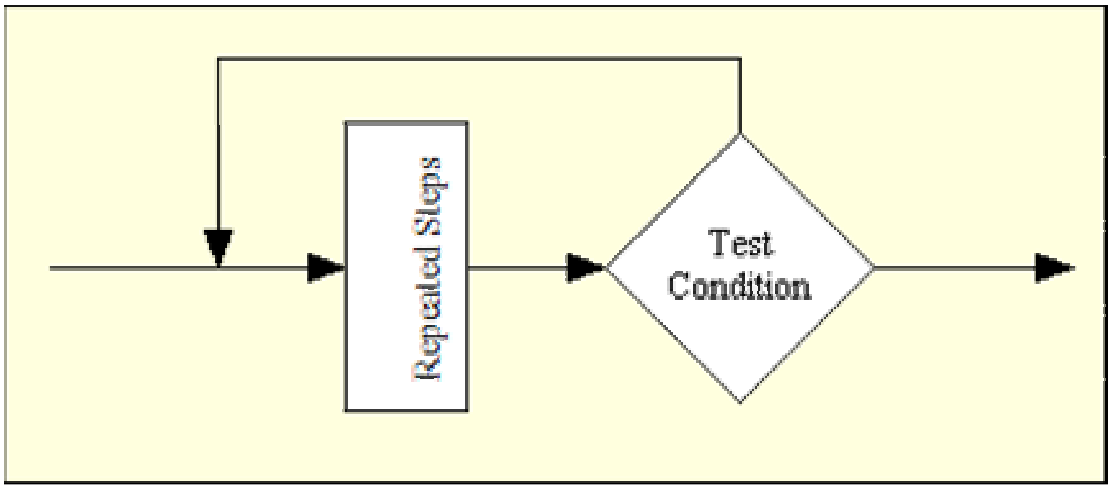
This is also known as a *conditional* construct because the program flow is dependent on the result of a test condition.



Structured Programming

Loops:

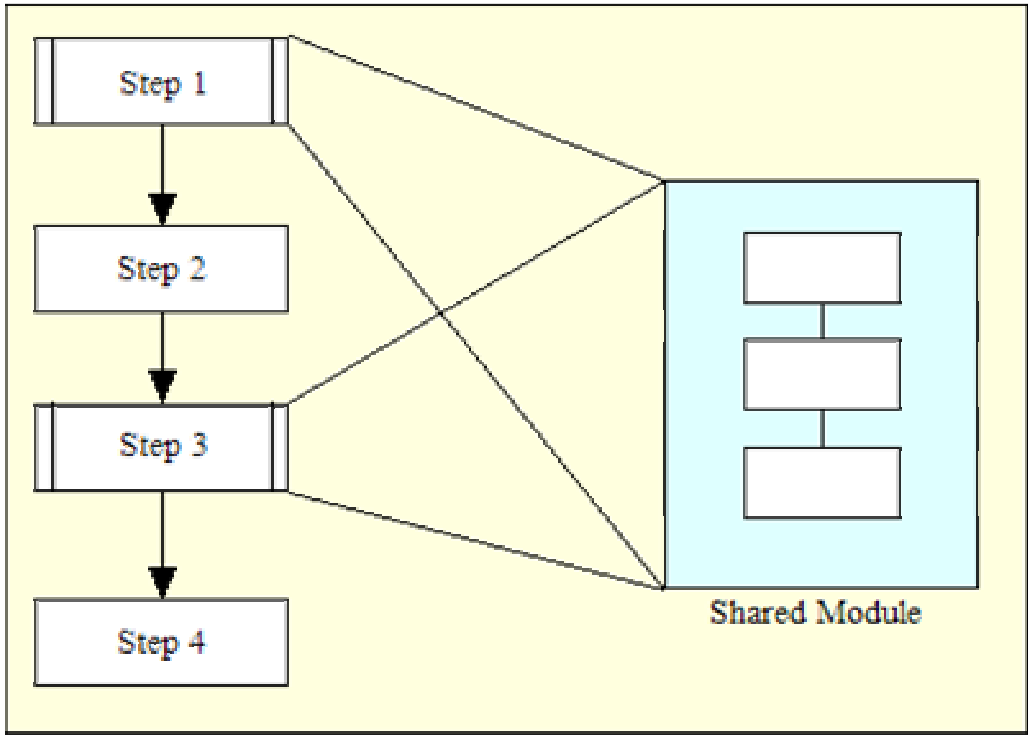
In this construct the program steps are repeated continuously until some test condition is reached, at which point control then flows past the loop into the next piece of program logic.



Structured Programming

Modules:

Here the program performs an identical sequence of actions several times. For convenience these common actions are placed in a module, which is a kind of mini-program which can be executed from within the main program. Other names for such a module are: *sub-routine*, *procedure* or *function*.



Structured Programming

Along with these structures programs also need a few more features to make them useful:

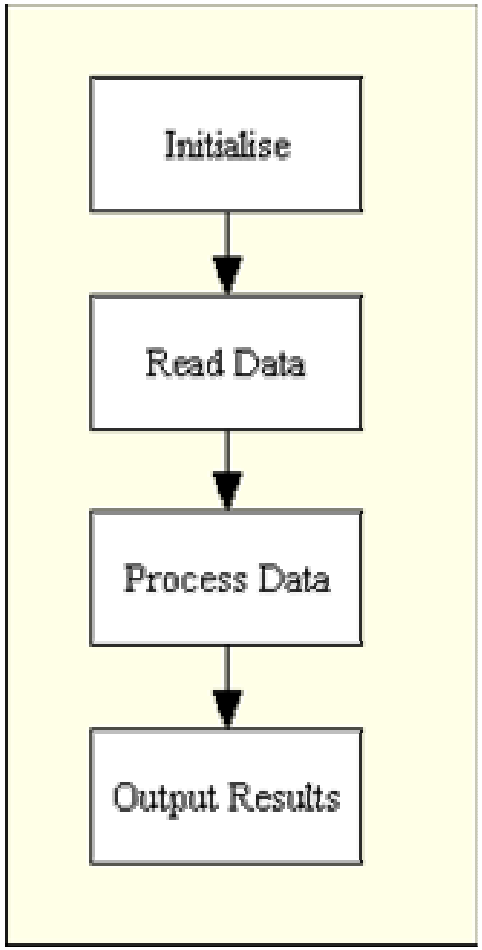
1) **Data (Raw Material)**

2) **Operations (add, subtract, compare etc.)**

3) **Input/Output capability (e.g. to display results)**

The Structure of a Program

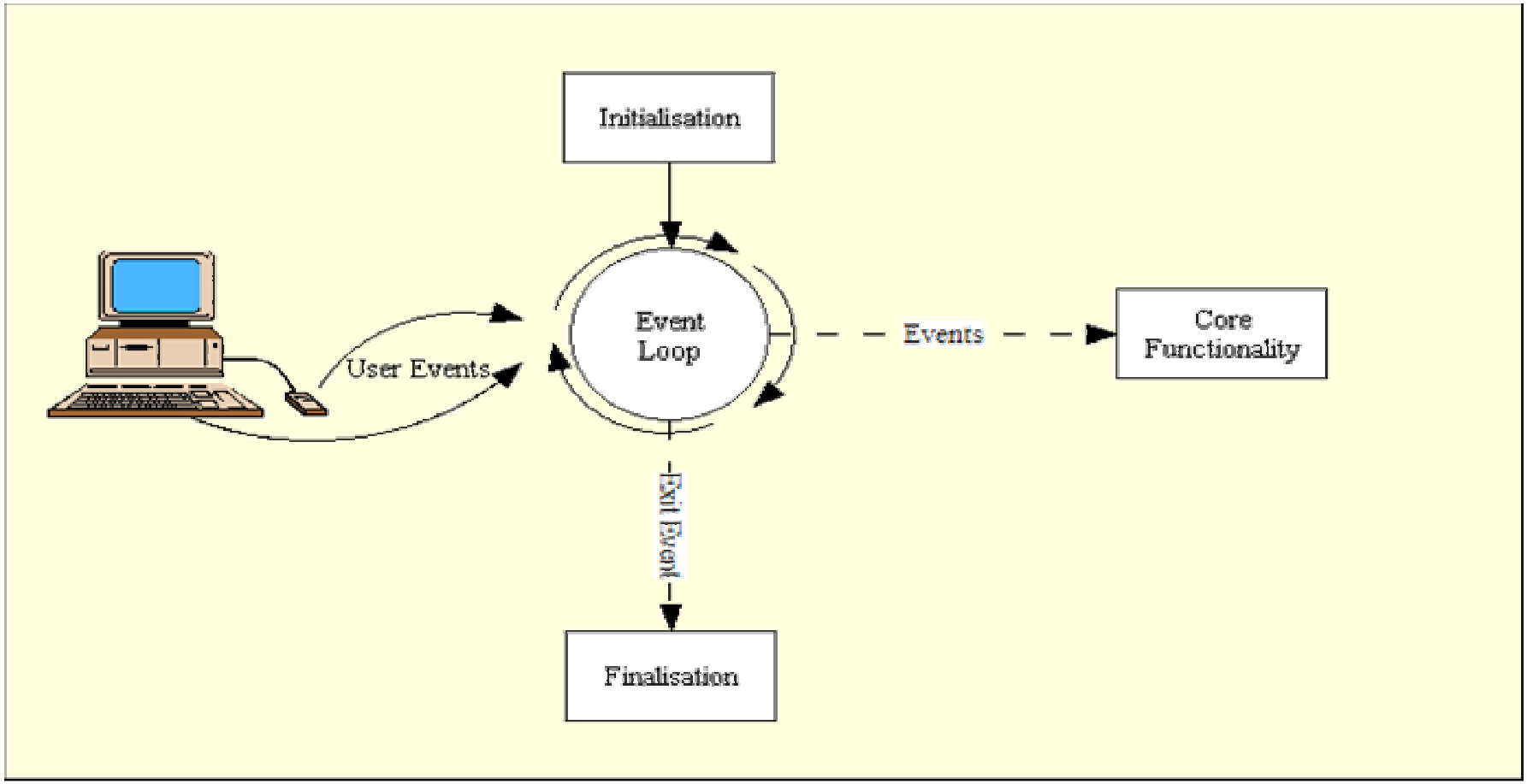
Batch Programs



The Structure of a Program

Event Driven Programs

Most GUI systems (and embedded control systems - like your Microwave, camera etc) are event driven. That is the operating system sends events to the program and the program responds to these as they arrive. Events can include things a user does - like clicking the mouse or pressing a key - or things that the system itself does like updating the clock or refreshing the screen.



Points to Remember

- **Programs control the computer**
- **Programming languages allow us to 'speak' to the computer at a level that is closer to how humans think**
- **Programs *operate* on *data***
- **Programs can be either *Batch oriented* or *Event driven***

Introduction to C

C is a general-purpose, structured programming language.

C is used for systems programming (e.g. for writing operating systems) as well as for applications programming (e.g. to solve a complicated system of mathematical equations, or for writing a program to bill customers).

C is characterized by the ability to write very concise programs, though actual implementations include extensive library functions. Users can write additional library functions of their own.

C programs are highly portable.

Introduction to C

C is a flexible, high-level, structured programming language.

C includes certain low-level features that are normally available only in assembly or machine language.

Programs written in C compile into small object programs that execute efficiently.

C is widely available.

C is largely machine-independent. Programs written in C are easily ported from one computer to another.

A Brief History of C

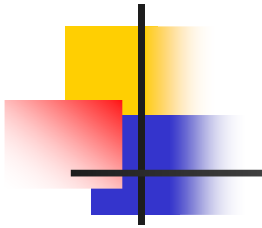
The C programming language was developed at Bell Labs during the early 1970's.

It derived from a computer language named B and from an earlier language BCPL.

Initially designed as a system programming language under UNIX it expanded to have wide usage on many different systems.

The earlier versions of C became know as K&R C after the authors of an earlier book, "The C Programming Language" by Kernighan and Ritchie.

A Brief History of C



As the language further developed and standardized, a version know as ANSI (American National Standards Institute) C became dominant.

Although it is no longer the language of choice for most new development, it still is used for some system and network programming as well as for embedded systems. More importantly, there is still a tremendous amount of legacy software still coded in this language and this software is still actively maintained.

A Note to Students

The complier turns the source code files, which are the text you will write, into a form that can be read and executed by a computer.

Compilers

C is a compiled language.

The C compiler is a program that reads source code, which is the C code written by a programmer, and produces an executable or binary file that in a format that can be read and executed (run) by a computer.

The source file is a plain text file containing your code. The executable file consists of machine code, 1's and 0's that are not meant to be understood or read by people, but only by computers.

Structure of a C Program

Every C program consists of one or more functions, one of which must be called main. The program will begin by executing the main function.

Each function must contain:

1. **A function heading – function name & arguments in ()**
2. **A list of argument declarations, if arguments are included**
3. **A compound statement – which comprises the remainder of the function.**
4. **Comments remarks may appear anywhere within the program, as long as they are placed within delimiters /* and */**

First Program - Hello World

```
#include <stdio.h>
void main()
{
        printf("Hello World \n");
}
```

Line 1: #include <stdio.h>

As part of compilation, the C compiler runs a program called the C preprocessor. The preprocessor is able to add code to your source file. In this case, the directive #include tells the preprocessor to include code from the file stdio.h. This file contains declarations for functions that the program needs to use. A declaration for the printf function is in this file.

First Program - Hello World

Line 2: void main()

This statement declares the main function. A C program can contain many functions but must always have one main function. A function is a self-contained module of code that can accomplish some task. The "void" specifies the return type of main. In this case, nothing is returned to the operating system.

Line 3: {

This opening bracket denotes the start of the program.

First Program - Hello World

Line 4: printf("Hello World \n");

Printf is a function from a standard C library that is used to print strings to the standard output, normally your screen. The compiler links code from these standard libraries to the code you have written to produce the final executable. The "\n" is a special format modifier that tells the printf to put a line feed at the end of the line. If there were another printf in this program, its string would print on the next line.

Line 5: }

This closing bracket denotes the end of the program.

Variables

A variable is used to hold data within your program. A variable represents a location in your computer's memory.

You can put data into this location and retrieve data out of it.

Every variable has two parts, a name and a data type.

Variable Names

Valid names can consist of letters, numbers and the underscore, but may not start with a number.

A variable name may not be a C keyword such as if, for, else, or while. Variable names are case sensitive. So, Age, AGE, aGE and AgE could be names for different variables, although this is not recommended since it would probably cause confusion and errors in your programs.

int, float and double are built in C data types

Variable Names

Which of the following are valid variable names?

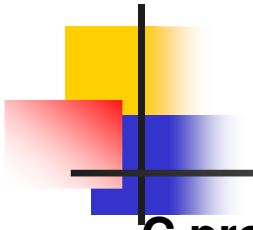
```
int idnumber;
int transaction_number;
int __my_phone_number__;
float 4myfriend;
float its4me;
double VeRyStRaNgE;
float while;
float myCash;
int CaseNo;
int CASENO;
int caseno;
```

Variable Names

Answers

`int idnumber;`	*Valid*
`int transaction_number;`	*Valid*
`int __my_phone_number__;`	*Valid*
`float 4myfriend;`	*Not valid,* variable names cannot start with a number
`float its4me;`	*Valid*
`double VeRyStRaNgE;`	*Valid*
`float while;`	*Not valid,* "while" is a keyword
`float myCash;`	*Valid*
`int CaseNo;`	These three are valid variable names. However, it is recommended to avoid using names that differ only in case since this practice leads to confusion and program errors.
`int CASENO;`	
`int caseno;`	

Data Types



C provides built in data types for character, float and integer data.

A mechanism, using the keyword typedef, exists for creating user-defined types.

Integer variables are used to store whole numbers. There are several keywords used to declare integer variables, including int, short, long, unsigned short, unsigned long.

The difference deals with the number of bytes used to store the variable in memory, long vs. short, or whether negative and positive numbers may be stored, signed vs. unsigned.

Examples:

```
int count;

int number_of_students = 30;
```

Data Types

Float variables are used to store floating point numbers.

Floating point numbers may contain both a whole and fractional part, for example, 52.7 or 3.33333333.

There are several keywords used to declare floating point numbers in C including float, double and long double. The difference here is the number of bytes used to store the variable in memory. Double allows larger values than float. Long double allows even larger values.

Examples:

float owned = 0.0;

float owed = 1234567.89;

Data Types

Character variables are used to store character values.

Character variables are declared with the keyword char.

Examples:

```
char firstInitial = 'J';

char secondInitial = 'K';
```

Constants

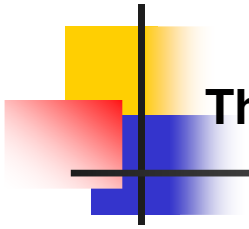
A constant is similar to a variable in the sense that it represents a memory location.

It differs, in that it cannot be reassigned a new value after initialization.

In general, constants are a useful feature that can prevent program bugs and logic errors.

Unintended modifications are prevented from occurring. The compiler will catch attempts to reassign new values to constants.

Constants



There are three techniques used to define constants in C.

Using #define

First, constants may be defined using the preprocessor directive #define.

The preprocessor is a program that modifies your source file prior to compilation.

Common preprocessor directives are #include, which is used to include additional code into your source file, #define, which is used to define a constant and #if/#endif, which can be used to conditionally determine which parts of your code will be compiled.

Constants

The #define directive is used as follows.

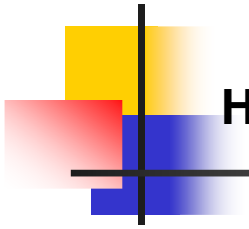
```
#define pi 3.1415
#define id_no 12345
```

Wherever the constant appears in your source file, the preprocessor replaces it by its value.

So, for instance, every "pi" in your source code will be replaced by 3.1415.

The compiler will only see the value 3.1415 in your code, not "pi". Every "pi" is just replaced by its value.

Constants



Here is a simple program illustrating the preprocessor directive

```
#include <stdio.h>
#define monday 1
#define tuesday 2
#define wednesday 3
#define thursday 4
#define friday 5
#define saturday 6
#define sunday 7
int main()
{
   int today = monday;
   if ((today == saturday) || (today == sunday))
   {
       printf("Weekend\n");
   }
   else
   {
      printf("Go to work or school\n");
   }
   return 0;
}
```

Constants

Using const variables

The second technique is to use the keyword const when defining a variable. When used the compiler will catch attempts to modify variables that have been declared const.

```
const float pi = 3.1415;
const int id_no = 12345;
```

There are two main advantages over the first technique. First, the type of the constant is defined. "pi" is float. "id_no" is int. This allows some type checking by the compiler. Second, these constants are variables with a definite scope. The scope of a variable relates to parts of your program in which it is defined. Some variables may exist only in certain functions or in certain blocks of code. You may want to use "id_no" in one function and a completely unrelated "id_no" in your main program.

Constants

Using enumerations

The third technique to declare constants is called enumeration.

An enumeration defines a set of constants.

In C an enumerator is of type int.

The enumeration is useful to define a set of constants instead of using multiple #defines.

Constants

```
#include <stdio.h>

enum days {monday=1,tuesday,wednesday,thursday,friday,saturday,sunday};

int main()
{
    enum days today = monday;
    if ((today == saturday) || (today == sunday))
    {
        printf("Weekend\n");
    }
    else
    {
        printf("Go to work or school\n");
    }
    return 0;
}
```

Constants

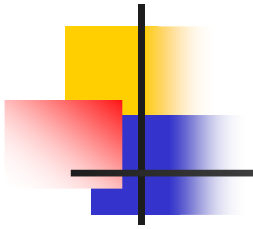
Example :
enum COLOR { RED, BLUE, GREEN};
enum SHAPE {SQUARE, RECTANGLE, TRIANGLE, CIRCLE, ELLIPSE};

Each enumerated constant (sometimes called an enumerator) has an integer value. Unless specified, the first constant has a value of zero. The values increase by one for each additional constant in the enumeration. So, RED equals 0, BLUE equals 1, and GREEN = 2. SQUARE equals 0, RECTANGLE equals 1, TRIANGLE equals 2 and so forth. The values of each constant can also be specified.

enum SHAPE
{SQUARE=5,RECTANGLE,TRIANGLE=17,CIRCLE,ELLIPSE};

Here, SQUARE equals 5, RECTANGLE equals 6, TRIANGLE equals 17, CIRCLE equals 18 and ELLIPSE equals 19.

Example – Area of a Circle



```
#include <stdio.h>                                  /* LIBRARY FILE ACCESS */
/* program to calculate area of a circle */        /* TITLE (COMMENT) */
main()                                             /* FUNCTION HEADING */
{
float radius, area;                                /* VARIABLE  DECLARATIONS*/
printf("Radius = ? ");                             /* OUTPUT STATEMENT(PROMPT) */
scanf("%f", &radius);                              /* INPUT STATEMENT */
area = 3.14159 * radius * radius;                  /* ASSIGNMENT STATEMENT */
Printf("Area = %f",area);                          /* OUTPUT STATEMENT */
}
```

Example – Area of a Circle

Execution of this program:

Radius = ? 3

Area = 28.274309

Desirable Program Characteristics

Important characteristics of well-written computer programs:

1. **Integrity – accuracy of calculations**
2. **Clarity - readability of the program**
3. **Simplicity - simple and straight forward program structure**
4. **Efficiency – execution speed and efficient memory utilization**
5. **Modularity – modular programming design**
6. **Generality – design a program to read in the values of certain key parameters rather than placing fixed values in the program**

Now Let us start our Computers

and get going