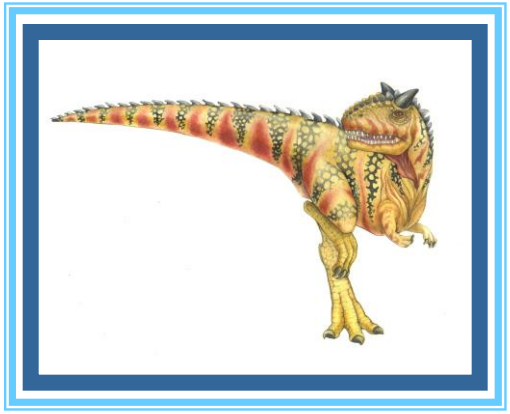
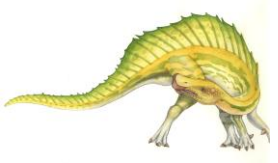
C Programming – Part A



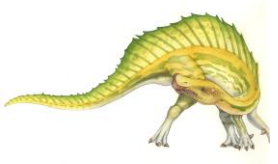
C Programming – Part A

- Introduction to C
- Input and Output
- Fundamental Data Types
- Variables and Constants
- Flow of Control
- Pointers
- Functions



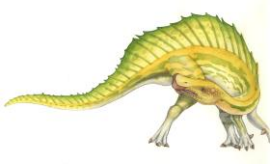
Introduction to C

- It is a general-purpose procedural programming language.
- It was developed by Dennis Ritchie between 1969 and 1973.
- It has facilities for structured programming.
- Its design provides constructs that map efficiently to typical machine instructions.
- A C program is a collection of functions.



Input and Output

- The function `printf()` is used for printing formatted output.
- The function `scanf()` is used for reading formatted input.
- These functions are in the standard library `stdio.h`.



printf()

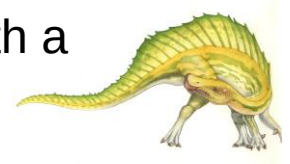
- It is an output function.
- It outputs a character stream to the standard output file `stdout`, which is normally connected to the screen.
- It takes an argument list: the first argument is called *control string* and the rest is called *other arguments.*
- Example:

```
printf("She sells %d %s for $%f.",  10, "apples",
9.99);
```

- Characters in the control string that are not part of a format specification are placed directly in the output stream; characters in the control string that are format specifications are replaced with the value of the corresponding argument in the other arguments.
- The output from the `printf()` function in the above example is:

```
She sells 10 apples for $9.990000.
```

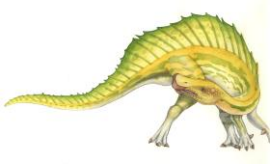
- A format specification is a string that begins with % and ends with a conversion character.



scanf()

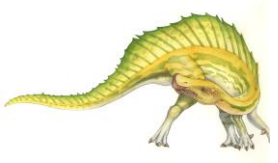
- It is an input function.
- It reads characters from the standard input file `stdin`, which is normally connected to the keyboard.
- It takes an argument list: the first argument is called *control string* and the rest is called *other arguments,* which are typically comma-separated pointer expressions.
- Example:

```
char a, b, c, s[100];
int n;
double x;
scanf("%c%c%c%d%s%lf", &a, &b, &c, &n, s, &x);
```



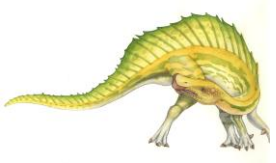
A “Hello World” C Program

```c
/*
// The traditional first program in honor of
// Dennis Ritchie, who invented C while
// at Bell Labs in 1972.
*/
#include <stdio.h>
int main(void)
{
   printf("Hello, world!\n");
   return 0;
}
```



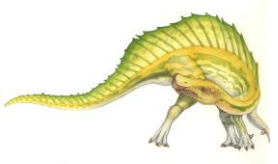
Fundamental Data Types

- Integral types
- Floating types
- Character type
- The `sizeof` operator



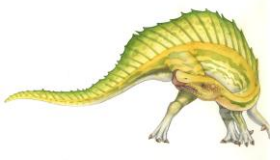
Integral Types

- Unsigned integers
  - `unsigned short, unsigned, unsigned long`
  - Nonnegative whole numbers: 0, 1, 2, 3, ….
  - Represented in 2, 4, or 8 bytes
- Signed integers
  - `short, int, long`
  - Whole numbers: …-3, -2, -1, 0, 1, 2, 3…
  - Memory requirements are the same as for unsigned integers, 2, 4, or 8 bytes



Floating Types

- `float` – single precision floating type
  - Represented in 4 bytes
- `double` – double precision floating type
  - Represented in 8 bytes
- `long double` – extended precision floating type
  - Represented in 16 bytes



Character Type

- The type `char` is used to represent characters.
- Example:

```
char c = 'a';
```

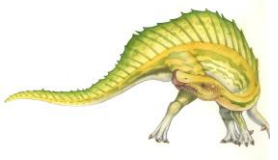
- A character is stored in one byte of memory.

  `'a'` is stored as `01100001`

Character Type

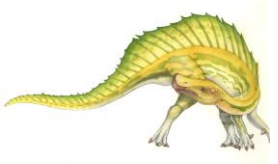
- Variables of any integral type can be used to represent characters.
- Characters are treated as small integers and, conversely, small integers are treated as characters.
- Example:

```
char c = 'a';     /* 'a' has ASCII encoding 97 */
int i = 65;       /* 65 is ASCII encoding for 'A' */
printf("%c", c + 1);    /* b is printed */
printf("%d", c + 2);    /* 99 is printed */
printf("%c", i + 3);    /* D is printed */
```



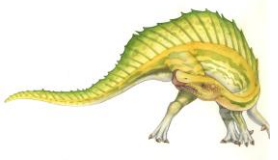
Character Type

```
/* Capitalize lowercase letters and double space */
#include <stdio.h>
#include <ctype.h>
int main(void)
{
   int   c;
   while ((c = getchar()) != EOF)
      if (islower(c))
       putchar(toupper(c));
      else if (c == '\n') {
         putchar('\n');
         putchar('\n');
      }
      else
         putchar(c);
   return 0;
}
```



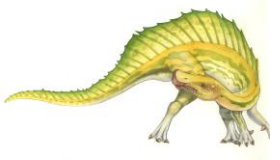
The `sizeof` Operator

- C provides an operator `sizeof` to find the number of bytes needed to store an object.
- Syntax:
  - `sizeof(type/object)`
- Examples:
  - `unsigned n = sizeof(char);`
  - `unsigned n = sizeof(a + 7.7);`
- The `sizeof` operator is system-dependent.



The `sizeof` Operator

```c
/* Compute the size of some fundamental types. */
#include <stdio.h>
int main(void)
{
   printf("\n");
   printf("Here are the sizes of some fundamental types:\n\n");
   printf("       char:%3d byte \n", sizeof(char));
   printf("      short:%3d bytes\n", sizeof(short));
   printf("        int:%3d bytes\n", sizeof(int));
   printf("       long:%3d bytes\n", sizeof(long));
   printf("   unsigned:%3d bytes\n", sizeof(unsigned));
   printf("      float:%3d bytes\n", sizeof(float));
   printf("     double:%3d bytes\n", sizeof(double));
   printf("long double:%3d bytes\n", sizeof(long double));
   printf("\n");
   return 0;
}
```



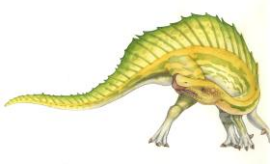
Variables

- Variables are used to stores data.
- Since different types of data have different sizes, the type of each variable must be specified.
- In C every variable must be declared before it can be used in a C program.
- Variables declaration:

```
char c;
int i, j;
```

- Variables initialisation:

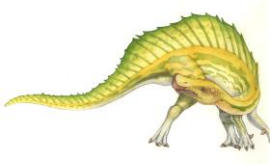
```
char c = 'a';
int i = 9;
```



Constants

- Literals
- Typed constant expressions
- Preprocessor definitions

```
int main ()
{
      double r = 5.0; // radius
      double area;
      area = r * r * 3.14159; //literal
      return 0;
}
```

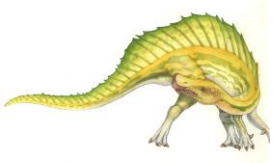


```
const double PI = 3.14159;  //typed constant

int main ()
{
        double r = 5.0; // radius
        double area;
        area = r * r * PI;
        return 0;
}
```

```
#define PI 3.14159  //pre-processor definition
int main ()
{
      double r = 5.0; // radius
      double area;
      area = r * r * PI;
      return 0;
}
```

Flow of Control

- The `if` and `if-else` statements
- The `while` statement
- The `for` statement
- The `do` statement
- The `switch` statement
- The `break` and `continue` statements



Pointers

- A pointer is a variable used to store a memory address.
- A pointer can be used to access memory and manipulate an address.
- Pointer declaration:

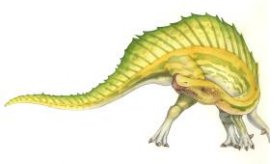
```
type *pointer;
```

- Pointer declaration example:

```
int *p;
```

- Pointer assignment:

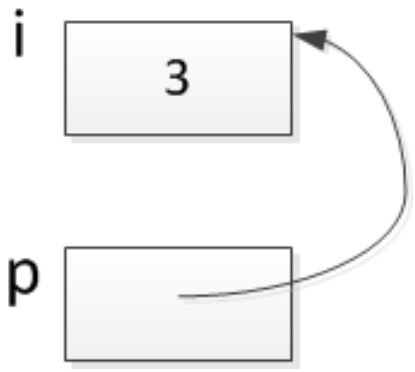
```
int *p;
p = 0;
p = NULL;
p = (int *) 1307;
```

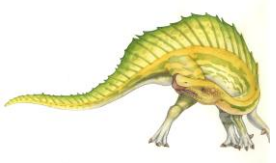


Pointer Operations

- Addressing operator **&** - to get the address of a variable

```
int i = 3, *p;
p = &i;
```

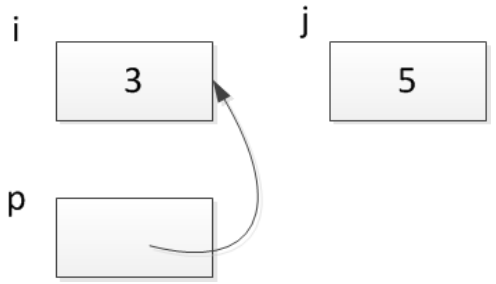




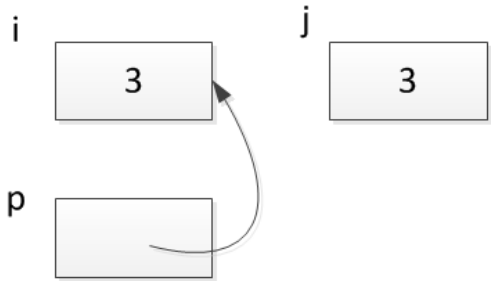
Pointer Operations – cont.

- Dereferencing operator ***** - to access the value stored at the location pointed by a pointer

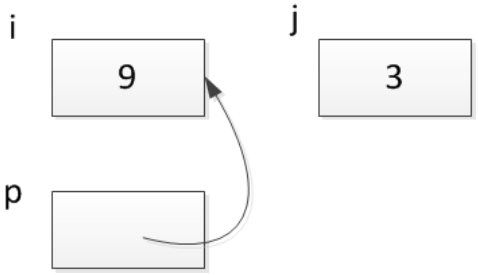
```
int i = 3, j = 5, *p;
p = &i;
```

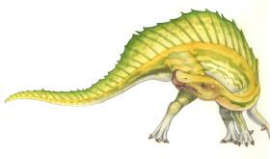



```
j = *p;
```




```
*p = 9;
```





Pointer to `void`

- Pointers to `void` are generic pointers, which can point to any data types.
- Pointers to `void` declaration:

```
void *pointer;
```

- An example of pointers to `void` declaration:

```
void *v;
```

- In ANSI (American National Standards Institute) C, an pointer can be assigned to another only when they have the same type, or when one of them is of type pointers to `void`.

Pointers – cont.

- Example:

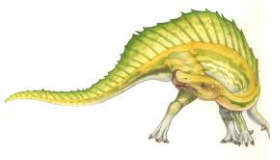
```
int *p1, *p2;
double *q;
void *v;
```

Legal assignments:

```
p1 = 0;
p1 = (int *) 1;
p1 = p2;
v = q;
p2 = v;
p1 = (int *) q;
```

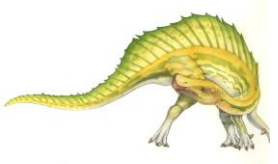
Illegal assignments:

```
p1 = q;
p2 = 1;
```



Introduction to Functions

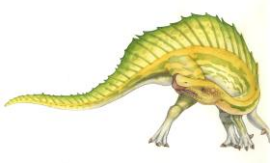
- Procedural programming is a programming methodology.
- C is a procedural programming language.
- A procedure in C is a *function*.
- The function construct in C is used to write code that solves a (small) problem.



Function Definition

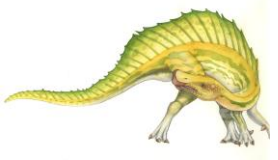
- The code that describes what a function does and how it does.
- Syntax:

```
return_type function_name(parameter_list)
{
    declarations
    statements
}
```



Function Invocation

- A procedural C program is made up of one or more functions, one of them being `main()`.
- The execution of a C program always begins with `main()`.
- When program control encounters a function name followed by parentheses, the function is invoked.
- A function is invoked by writing its name and an appropriate list of arguments within parentheses.
- Typically, the arguments match in number and type the parameters in the parameter list in the function definition.
- When a function is invoked, the function has program control.
- After the function finishes its work, program control is returned to the calling function and the program continues to execute.



```
#include <stdio.h>
void   prn_message(const int k);

int main(void)
{
   int   n;

   printf("There is a message for you.\n”);
   printf("How many times do you want to see it? \n”);
   scanf(“%d”, &n);
   prn_message(n);
   return 0;
}

void prn_message(const int  k)
{
   printf("Here is the message:\n”);
   for (int i = 0; i < k; ++i)
      printf("Have a nice day!\n”);
}
```

function prototype

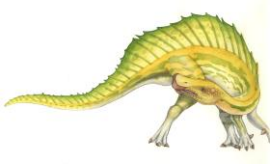
A function **parameter** is a variable used in the prototype and the definition of a function.

A function **argument** is a value passed to a function in the place of a parameter in the function invocation.

The `return` statement

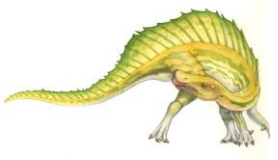
- When a `return` statement is executed, program control is immediately returned to the calling function.
- If an expression follows the keyword `return`, the value of the expression is also returned to the calling function.
- Two forms of `return` statement:
  - `return; // used for void function`
  - `return expression; // used for non-void function`
- Examples:

```
return;
return 1;
return i++;
return (a + b);
```



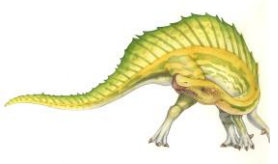
The `return` statement – cont.

- If a function does not contain a `return` statement, after finishing executing the code in the function body, the control will still return to the calling function as if there were an implicit `return` statement at the end of the function body.
  - But, it is a good practice to explicitly put a `return` statement at the end of the function body.



Call-by-Value

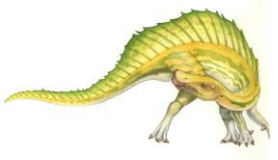
- When variables are passed as arguments to a function, their values are copied to the corresponding parameters in the parameter list of the function.
- The values of the variables in the calling function will remain unchanged.



```c
#include <stdio.h>

int   min(int a, int b);  //function prototype

int main(void)
{
        int   j, k, m;
        printf("Input two integers:  ");
        scanf("%d%d", j, k);
        m = min(j, k);
        printf("The smaller is %d\n", m);
        return 0;
}

int min(int a, int b)
{
        if (a < b)
                return a;
        else
                return b;
}
```

Call-by-Reference

- The addresses of variables are used in the parameter list of a function definition, and the addresses of variables are passed to their corresponding parameters.
- The values of the variables in the calling function will be changed by the function.



```c
#include <stdio.h>

void swap(int *p, int *q);

int main(void)
{
        int   a = 3, b = 7;
        printf("%d  %d\n", a, b); /* 3  7 printed */
        swap(&a, &b);
        printf("%d  %d\n", a, b); /* 7  3 printed */
        return 0;
}

void swap(int *p, int *q)
{
        int   tmp;
        tmp = *p;
        *p = *q;
        *q = tmp;
}
```

Acknowledgement

- Examples used in this lecture are from
  - A. Kelly and I. Pohl, *C by Dissection – The Essentials of C Programming*, 4th Edition, Addison Wesley, 2001.