**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **18/June/2020** | **Name:** | **Prashantha naik** |
| **Course:** | **C programming** | **USN:** | **4al17ec074** |
| **Topic:** | **1.Module 1: Basic Concept**  **2.** **Module 2: Conditionals & Loops** | **Semester & Section:** | **6th b** |
| **GitHub Repository:** | **prashanth\_course** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report – Report can be typed or hand written for up to two pages.**  **Basic Concept**  Introducing C C is a general-purpose programming language that has been around for nearly 50 years. C has been used to write everything from operating systems (including Windows and many others) to complex programs like the Python interpreter, Git, Oracle database, and more. The versatility of C is by design. It is a low-level language that relates closely to the way machines work while still being easy to learn.  **Data Types**  C supports the following basic data types: int: integer, a whole number. float: floating point, a number with a fractional part. double: double-precision floating point value. char: single character.  **input**  C supports a number of ways for taking user input. get char() Returns the value of the next single character input.  For example:  #include <stdio.h>  int main() { char a = getchar();  printf("You entered: %c", a);  return 0; }  The input is stored in the variable a.  The gets() function is used to read input as an ordered sequence of characters, also called a string. A string is stored in a char array. For example:  #include <stdio.h>  int main() { char a[100];  gets(a);   printf("You entered: %s", a);  return 0; }  **Comments**  Comments are explanatory information that you can include in a program to benefit the reader of your code. The compiler ignores comments, so they have no affect on a program. A comment starts with a slash asterisk /\* and ends with an asterisk slash \*/ and can be anywhere in your code. Comments can be on the same line as a statement, or they can span several lines.  **Arithmetic Operators** C supports arithmetic operators + (addition), - (subtraction), \* (multiplication), / (division), and % (modulus division). Operators are often used to form a numeric expression such as 10 + 5, which in this case contains two operands and the addition operator.  **Module 2: Conditionals & Loops**  **Conditionals**  Conditionals are used to perform different computations or actions depending on whether a condition evaluates to true or false.  The if Statement  The if statement is called a conditional control structure because it executes statements when an expression is true. For this reason, the if is also known as a decision structure  **The switch Statement**  The switch statement often provides a more elegant solution to if-else if and nested if statements.  The switch takes the form:  switch (expression) {  case val1:  statements  break;  case val2:  statements  break;  default:  statements  }  **The while Loop**  The while statement is called a loop structure because it executes statements repeatedly while an expression is true, looping over and over again. It takes the form:  while (expression) {  statements  }  The expression evaluates to either true or false, and statements can be a single statement or, more commonly, a code block enclosed by curly braces { }.  For example:  #include <stdio.h>  int main() {  int count = 1;    while (count < 8) {  printf("Count = %d\n", count);  count++;  }    return 0;  }  **The for Loop**  The for statement is a loop structure that executes statements a fixed number of times.  for (initvalue; condition; increment) {  statements;  }  The initvalue is a counter set to an initial value. This part of the for loop is performed only once.  For example, the program below displays 0 through 9:  int i;  int max = 10;    for (i = 0; i < max; i++) {  printf("%d\n", i);  } |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **18/June/2020** | **Name:** | **Prashantha naik** | |
| **Course:** | **C programming** | **USN:** | **4al17ec074** | |
| **Topic:** | **1.Module 3: Functions, Array & Pointers.**  **2.** **Module 4: Strings & Function Pointers** | **Semester&Section:** | **6th b** | |
| **Git hub repository** | **prashanth\_couse** |  |  | |
| **AFTERNOON SESSION DETAILS** | | | |
| **Image of session** | | | |
| **Report – Report can be typed or hand written for up to two pages.**  **Functions, Array & Pointers.**  **Functions in C**  Functions are central to C programming and are used to accomplish a program solution as a series of subtasks.  By now you know that every C program contains a main() function. And you're familiar with the printf() function.  You can also create your own functions.  A function:  • is a block of code that performs a specific task  • is reusable  • makes a program easier to test  • can be modified without changing the calling program  Even a simple program is easier to understand when main() is broken down into subtasks that are implemented with functions.  For example, it's clear that the goal of this program is to calculate the square of a number:  int main() {  int x, result;    x = 5;  result = square(x);  printf("%d squared is %d\n", x, result);    return 0;  }  **Arrays in C**  An array is a data structure that stores a collection of related values that are all the same type.  Arrays are useful because they can represent related data with one descriptive name rather than using separate variables that each must be named uniquely.  For example, the array test\_scores[25] can hold 25 test scores.  An array declaration includes the type of the values it stores, an identifier, and square brackets [ ] with a number that indicates the array size.  For example:  int test scores[25]; /\* An array size 25 \*/  Note that initial values are separated by commas and placed inside curly braces { }.  An array can be partially initialized, as in:  float prices[5] = {3.2, 6.55};  **Pointers and Arrays**  Pointers are especially useful with arrays. An array declaration reserves a block of contiguous memory addresses for its elements. With pointers, we can point to the first element and then use address arithmetic to traverse the array:  + is used to move forward to a memory location  - is used to move backward to a memory location  Consider the following program:  int a[5] = {22, 33, 44, 55, 66};  int \*ptr = NULL;  int i;  ptr = a;  for (i = 0; i < 5; i++) {  printf("%d ", \*(ptr + i));  }  Try It Yourself  The program output is: 22 33 44 55 66  An important concept with arrays is that an array name acts as a pointer to the first element of the array. Therefore, the statement ptr = a can be thought of as ptr = &a[0].  Consider the following statement, which prints the first element of the array: printf("%d ", \*a);  **Strings**  A string in C is an array of characters that ends with a NULL character '\0'.  A string declaration can be made in several ways, each with its own considerations.  For example:  char str\_name[str\_len] = "string";  This creates a string named str\_name of str\_len characters and initializes it to the value "string".  When you provide a string literal to initialize the string, the compiler automatically adds a NULL character '\0' to the char array.  For this reason, you must declare the array size to be at least one character longer than the expected string length.  The statements below creates strings that include the NULL character. If the declaration does not include a char array size, then it will be calculated based on the length of the string in the initialization plus one for '\0':  char str1[6] = "hello";  char str2[ ] = "world"; /\* size 6 \*/ | | | |