**자료구조론 CC343\_2207**

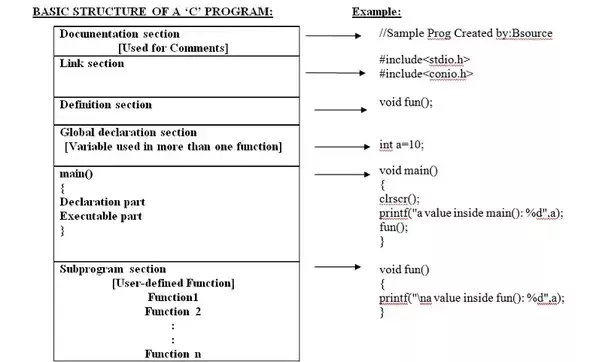
**Reading assignment 1**

**경기대학교 컴퓨터공학부**

**201511837 이상민**

**Review Questions**

1. Discuss the structure of a C program. (C 프로그램의 구조에 대해 논의합니다.)



* 1. **Documentation section** : The documentation section consists of a set of comment lines giving the name of the program, the author and other details, which the programmer would like to use later.
  2. **Link section** : The link section provides instructions to the compiler to link functions from the system library.
  3. **Definition section** : The definition section defines all symbolic constants.
  4. **Global declaration section** : There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions. This section also declares all the user-defined functions.
  5. **main () function section** : Every C program must have one main function section. This section contains two parts; declaration part and executable partDeclaration part : The declaration part declares all the variables used in the executable part.Executable part : There is at least one statement in the executable part. These two parts must appear between the opening and closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function is the logical end of the program. All statements in the declaration and executable part end with a semicolon.
  6. **Subprogram section** : The subprogram section contains all the user-defined functions that are called in the main () function. User-defined functions are generally placed immediately after the main () function, although they may appear in any order. Note:All section, except the main () function section may be absent when they are not required.

1. Differentiate between declaration and declaration (선언과 선언을 구분합니다.)

|  |  |
| --- | --- |
| DECLARATION | DEFINITION |
| A variable or a function can be declared any number of times | A variable or a function can be defined only once |
| Memory will not be allocated during declaration | Memory will be allocated |
| **Int f(int);**  The above is a function declaration. This declaration is just for informing the compiler that a function named f with return type and argument as int will be used in the function. | **Int f(int a)**  **{**  **Return a;**  **}**  The system allocates memory by seeing the above function definition. |

1. How is memory reserved using a declaration statement? (선언문을 사용하여 메모리를 어떻게 예약합니까?)

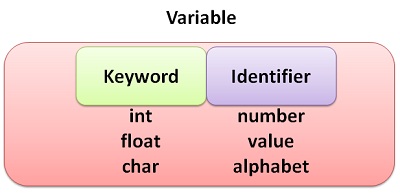
Memory is reserved using data type in the variable declaration. A programming language implementation has predefined sizes for its data types.

For example, in C# the declaration int i; will reserve 32 bits for variable i.

A pointer declaration reserves memory for the address or the pointer variable, but not for the data that it will point to. The memory for the data pointed by a pointer has to be allocated at runtime.

The memory reserved by the compiler for simple variables and for storing pointer address is allocated on the stack, while the memory allocated for pointer referenced data at runtime is allocated on the heap.

1. What do you understand by identifiers and keywords? (식별자와 키워드로 무엇을 이해합니까?)



Every language has keywords and identifiers, which are only understood by its compiler. Keywords are predefined reserved words, which possess special meaning. Each keyword defines the “type” declared data. Keywords should not be used as identifiers. An identifier is a unique name given to a particular variable, function or label of class in the program. To create a variable, both a keyword and an identifier are bind together.

1. Explain the terms variables and constants. How many types of variables are supported by C? (항 변수 및 상수를 설명합니다. C에서 지원하는 변수 유형은 몇 가지입니까?)

A variable is a named memory location which temporarily stores data that can change while the program is running.

A constant is a named memory location which temporarily stores data that remains the same throughout the execution of the program.

C++ recognizes eleven built-in data types which are designated by reserved words :

* 1. char
  2. short
  3. float
  4. long double
  5. unsigned int
  6. unsigned long
  7. int
  8. long
  9. double
  10. unsigned char
  11. unsigned short

1. What does the data type of a variable signify? (변수의 데이터 유형은 무엇을 의미합니까?)

The data type of a value (or variable in some contexts) is an attribute that tells what kind of data that value can have. Most often the term is used in connection with static typing of variables in programming languages like C/C++, Java and C# etc, where the type of a variable is known at compile time. Data types include the storage classifications like integers, floating point values, strings, characters etc.

Data types define particular characteristics of data used in software programs and inform the compilers about predefined attributes required by specific variables or associated data objects.

1. Write a short note on basic data types that the C language supports. (C 언어가 지원하는 기본 데이터 유형에 대해 짧은 메모를 작성합니다.)

Data types specify how we enter data into our programs and what type of data we enter. C language has some predefined set of data types to handle various kinds of data that we can use in our program. These datatypes have different storage capacities.

C language supports 2 different type of data types:

* 1. **Primary data types:**

These are fundamental data types in C namely integer(int), floating point(float), character(char) and void.

* 1. **Derived data types:**

Derived data types are nothing but primary datatypes but a little twisted or grouped together like array, stucture, union and pointer. These are discussed in details later.

Data type determines the type of data a variable will hold. If a variable x is declared as int. it means x can hold only integer values. Every variable which is used in the program must be declared as what data-type it is.

1. Why do we include <stdio.h> in our programs? (<stdio>를 포함시키는 이유는 무엇입니까? 우리 프로그램에서요?)

To include standard library functions

You write a program in C language or in any other programming language and you compile it , compiler changes code into assembly language and after that assembly language program changed into machine language (that 1 and 0 language).

When you write any program you use some predefined functions (e.g. Printf() or scanf() ), these functions are predefined, you are just using its functionalities but you didn't define it in your program. So if you want to use these predefined functions then you must give definitions of these functions in your program, so you are just including standard libraries in your program to give definitions of these functions. Later linker and loader will do their work and linked that library into code.

If definition of some functions is not available in stdio.h , and you want to use that function then you need to include anyother header files (e.g. : conic.h or math.h etc)

Basically If you don't want to define any function and that function is available in any library so you can use it but you have to include that particular library in your program.

1. What are header files? Explain their significance. (헤더 파일이란 무엇입니까? 그들의 중요성을 설명합니다.)

A header file is a library of a no. of functions, which could be used (once or more than once) in some or the other programs. Thus, instead of defining all functions separately we can assemble them in a single library, the header file. These can be predefined or user defined.Hence, it reduces the line of codes and also the complexity.

The include directive allows libraries of code to be developed which help to:

* + ensure that everyone uses the same version of a data layout definition or procedural code throughout a program.
  + easily cross-reference where components are used in a system.
  + easily change programs when needed (only one master copy to change).
  + save time by not needing to code extensive data layouts (minor, but useful).

1. Write short notes on printf and scanf functions. (printf 및 scanf 기능에 대한 짧은 메모를 작성합니다.)

printf() and scanf() functions are inbuilt library functions in C programming language which are available in C library by default. These functions are declared and related macros are defined in “stdio.h” which is a header file in C language.

**1. printf() function in C language:**

* + In C programming language, printf() function is used to print the “character, string, float, integer, octal and hexadecimal values” onto the output screen.
  + We use printf() function with %d format specifier to display the value of an integer variable.
  + Similarly %c is used to display character, %f for float variable, %s for string variable, %lf for double and %x for hexadecimal variable.
  + To generate a newline,we use “\n” in C printf() statement.

**2. scanf() function in C language:**

* In C programming language, scanf() function is used to read character, string, numeric data from keyboard
* Consider below example program where user enters a character. This value is assigned to the variable “ch” and then displayed.
* Then, user enters a string and this value is assigned to the variable “str” and then displayed.

1. Write a short note on operators available in C language. (C 언어로 제공되는 연산자에 대해 짧은 메모를 작성합니다.)

**Arithmetic operators**

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | adds two operands |
| - | subtract second operands from first |
| \* | multiply two operand |
| / | divide numerator by denominator |
| % | remainder of division |
| ++ | Increment operator - increases integer value by one |
| -- | Decrement operator - decreases integer value by one |

**Relational operators**

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | Check if two operand are equal |
| != | Check if two operand are not equal. |
| > | Check if operand on the left is greater than operand on the right |
| < | Check operand on the left is smaller than right operand |
| >= | check left operand is greater than or equal to right operand |
| <= | Check if operand on left is smaller than or equal to right operand |

**Logical operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | Logical AND | (a && b) is false |
| || | Logical OR | (a || b) is true |
| ! | Logical NOT | (!a) is false |

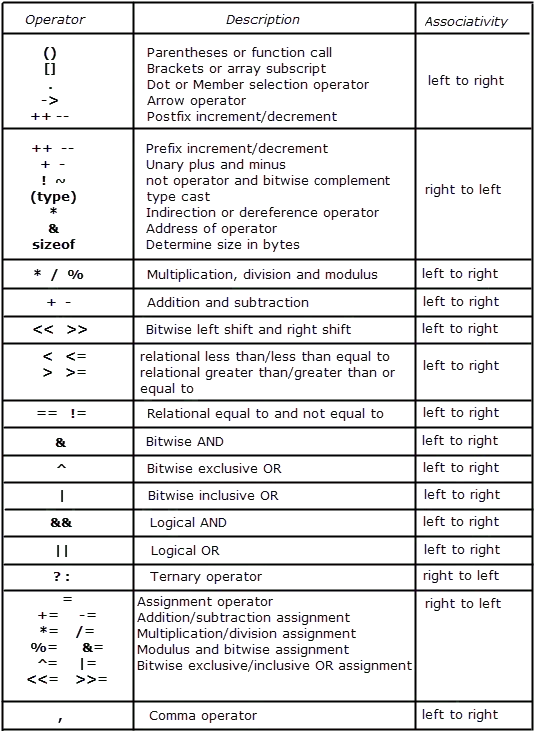
**Bitwise operators**

|  |  |
| --- | --- |
| **Operator** | **Description** |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise exclusive OR |
| << | left shift |
| >> | right shift |

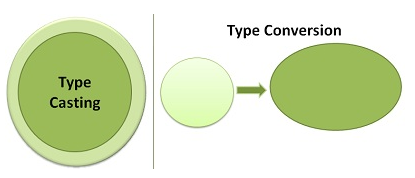
**Assignment Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | assigns values from right side operands to left side operand | a=b |
| += | adds right operand to the left operand and assign the result to left | a+=b is same as a=a+b |
| -= | subtracts right operand from the left operand and assign the result to left operand | a-=b is same as a=a-b |
| \*= | mutiply left operand with the right operand and assign the result to left operand | a\*=b is same as a=a\*b |
| /= | divides left operand with the right operand and assign the result to left operand | a/=b is same as a=a/b |
| %= | calculate modulus using two operands and assign the result to left operand | a%=b is same as a=a%b |

1. Draw the operator precedence chart. (연산자 우선 순위 차트를 그립니다.)



1. Differentiate between typecasting and type conversion. (유형캐스팅과 유형 변환을 구분합니다.)



The basic difference between type conversion and type casting, i.e. type conversion is made “automatically” by compiler whereas, type casting is to be “explicitly done” by the programmer.

The two terms “type casting” and “type conversion” occur when there is a need to convert one data type to another. When the two types are compatible with each other, then the conversion of one type to other is done automatically by the compiler.

1. What are decision control statements? Explain in detail. (의사결정 통제 진술이란 무엇입니까? 자세히 설명해 주세요.)

**WHAT ARE DECISION CONTROL STATEMENTS?**

The decision control statements are the decision making statements that decides the order of execution of statements based on the conditions. In the decision making statements the programmer specify which conditions are to be executed or tested with the statements to be executed if the condition is true or false.

**SOME OF THE DECISION CONTROL STATEMENTS ARE**

* if statement
* if-else statement
* nested if statements
* switch statement

Let us discuss all these statements briefly.

**IF STATEMENT**

The if statement consists a condition which is followed by one or some of the statements, if the condition is true then the statements will be executed or else not. This statement is the simple and easy decision control statement.

**LET US HAVE A LOOK AT THE SYNTAX :**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | if (condition)    {    //statements to be executed if condition is true    } |
|  |  |

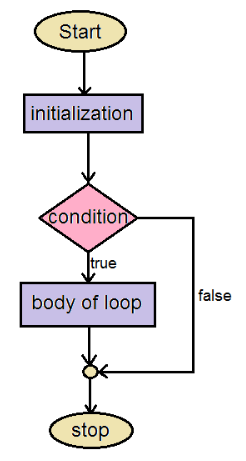
1. Write a short note on the iterative statements that C language supports. (C 언어가 지원하는 반복 문장에 짧은 메모를 작성합니다.)

Iteration is the process where a set of instructions or statements is executed repeatedly for a specified number of time or until a condition is met. These statements also alter the control flow of the program and thus can also be classified as control statements in C Programming Language.

Iteration statements are most commonly know as loops. Also the repetition process in C is done by using loop control instruction. There are three types of looping statements:

* For Loop
* While Loop
* Do-while loop

A loop basically consists of three parts: initialization, test expression, increment/decrement or update value



1. When will you prefer to work with a switch statement? (언제 스위치 문구를 사용하는 것이 좋습니까?)

Switch statement is useful when we are having multiple options for the same variable but when we have to use mutiple conditions simultaneously then it would be better to use if else.

1. Define function. Why are they needed? (기능을 정의합니다. 그들은 왜 필요합니까?)

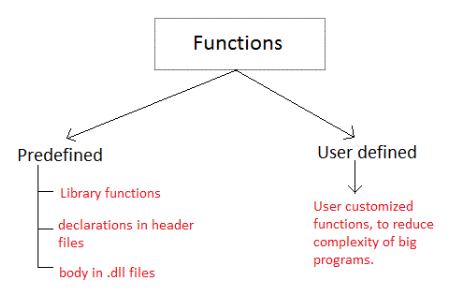
A function is a block of code that performs a particular task.

There are many situations where we might need to write same line of code for more than once in a program. This may lead to unnecessary repetition of code, bugs and even becomes boring for the programmer. So, C language provides an approach in which you can declare and define a group of statements once in the form of a function and it can be called and used whenever required.

These functions defined by the user are also know as User-defined Functions

C functions can be classified into two categories,

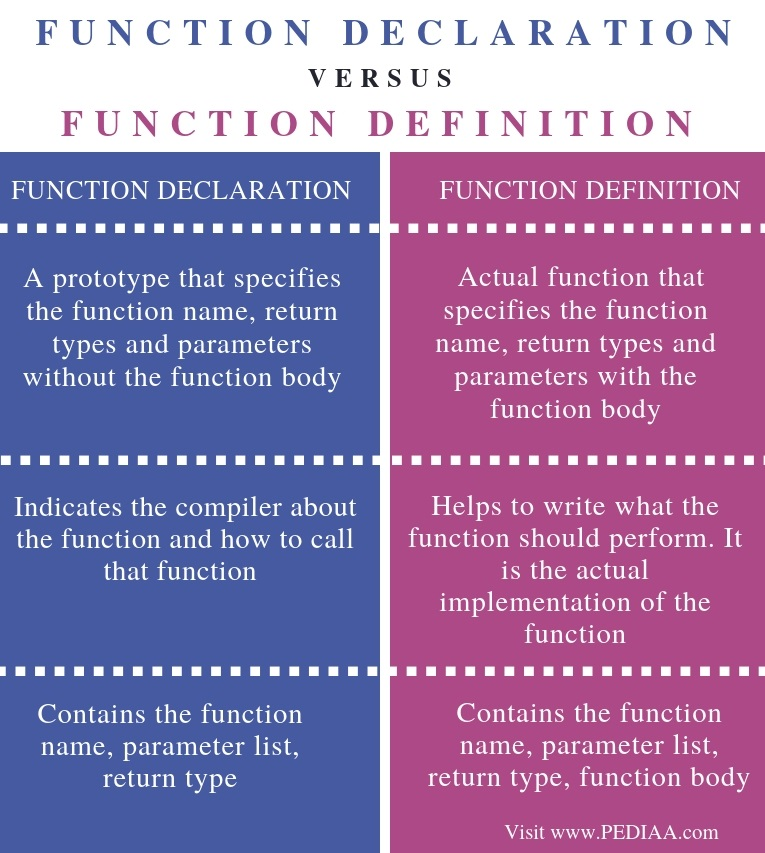
1. Library functions
2. User-defined functions



Library functions are those functions which are already defined in C library, example printf(), scanf(), strcat() etc. You just need to include appropriate header files to use these functions. These are already declared and defined in C libraries.

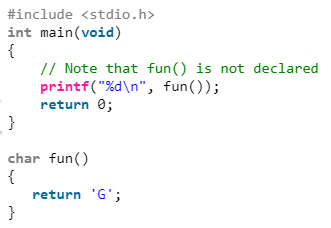
A User-defined functions on the other hand, are those functions which are defined by the user at the time of writing program. These functions are made for code reusability and for saving time and space.

1. Differentiate between function declaration and function definition. (함수 선언과 함수 정의를 구분합니다.)



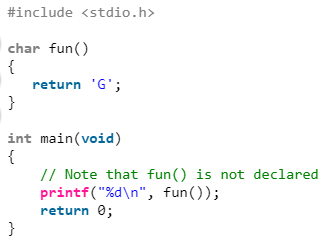
1. Why is function declaration statement placed prior to function definition? (기능 정의보다 기능 선언문이 먼저 배치되는 이유는 무엇입니까?)

For example, the following program fails in compilation.



if the function char fun() in above code is defined before main() then it will compile and run perfectly.

for example, the following program will run properly.



It is always recommended to declare a function before its use so that we don’t see any surprises when the program is run

1. Explain the concept of making function calls. (함수 호출의 개념을 설명합니다.)

As we know, a function is a set of instruction used to perform a specific task. Every c program must has a main() function, without a main() function, none of your programs can run.

A main() function is the head of the program. A main() function can call any function in the program, but there is no possibilities that any function can call the main() function.

There are 4 methods for calling a function :

* 1. Take something, return something
  2. Take something, return nothing
  3. Take nothing, return something
  4. Take nothing, return nothing

1. Differentiate between call by value and call by reference using suitable examples. (적절한 예를 사용하여 가치에 따라 통화와 참조를 구분합니다.)

**What is Call by Value method?**

Call by value method copies the value of an argument into the formal parameter of that function. Therefore, changes made to the parameter of the main function do not affect the argument.

In this parameter passing method, values of actual parameters are copied to function's formal parameters, and the parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of the caller.

**What is Call by Reference method?**

Call by reference method copies the address of an argument into the formal parameter. In this method, the address is used to access the actual argument used in the function call. It means that changes made in the parameter alter the passing argument.

In this method, the memory allocation is the same as the actual parameters. All the operation in the function are performed on the value stored at the address of the actual parameter, and the modified value will be stored at the same address.

**Call by Value vs. Call by Reference**

| **Parameters** | **Call by value** | **Call by reference** |
| --- | --- | --- |
| Definition | While calling a function, when you pass values by copying variables, it is known as "Call By Values." | While calling a function, in programming language instead of copying the values of variables, the address of the variables is used it is known as "Call By References. |
| Arguments | In this method, a copy of the variable is passed. | In this method, a variable itself is passed. |
| Effect | Changes made in a copy of variable never modify the value of variable outside the function. | Change in the variable also affects the value of the variable outside the function. |
| Alteration of value | Does not allow you to make any changes in the actual variables. | Allows you to make changes in the values of variables by using function calls. |
| Passing of variable | Values of variables are passed using a straightforward method. | Pointer variables are required to store the address of variables. |
| Value modification | Original value not modified. | The original value is modified. |
| Memory Location | Actual and formal arguments will be created in different memory location | Actual and formal arguments will be created in the same memory location |
| Safety | Actual arguments remain safe as they cannot be modified accidentally. | Actual arguments are not Safe. They can be accidentally modified, so you need to handle arguments operations carefully. |
| Default | Default in many programming languages like C++.PHP. Visual Basic NET, and C#. | It is supported by most programming languages like JAVA, but not as default. |

1. Write a short note on pointers. (포인터에 짧은 메모를 작성합니다.)

**C-Pointer**

* + Pointers in C language is a variable that stores/points the address of another variable. A Pointer in C is used to allocate memory dynamically i.e. at run time. The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.
  + Pointer Syntax : data\_type \*var\_name; Example : int \*p; char \*p;
  + Where, \* is used to denote that “p” is pointer variable and not a normal variable.

**KEY POINTS TO REMEMBER ABOUT POINTERS IN C:**

* Normal variable stores the value whereas pointer variable stores the address of the variable.
* The content of the C pointer always be a whole number i.e. address.
* Always C pointer is initialized to null, i.e. int \*p = null.
* The value of null pointer is 0.
* & symbol is used to get the address of the variable.
* symbol is used to get the value of the variable that the pointer is pointing to.
* If a pointer in C is assigned to NULL, it means it is pointing to nothing.
* Two pointers can be subtracted to know how many elements are available between these two pointers.
* But, Pointer addition, multiplication, division are not allowed.
* The size of any pointer is 2 byte (for 16 bit compiler).

1. Explain the difference between a null pointer and a void pointer. (null 포인터와 void 포인터의 차이를 설명합니다.)

Null pointer is a special reserved value of a pointer. A pointer of any type has such a reserved value. Formally, each specific pointer type (int \*, char \* etc.) has its own dedicated null-pointer value. Conceptually, when a pointer has that null value it is not pointing anywhere.

Void pointer is a specific pointer type - void \* - a pointer that points to some data location in storage, which doesn't have any specific type.

So, once again, null pointer is a value, while void pointer is a type. These concepts are totally different and non-comparable. That essentially means that your question, as stated, is not exactly valid. It is like asking, for example, "What is the difference between a triangle and a car?".

1. How are generic pointers different from other pointer variables? (일반 포인터는 다른 포인터 변수와 어떻게 다릅니까?)

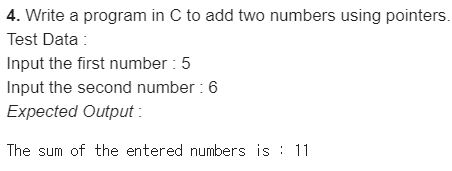
Generic pointer is like master placeholder that can hold address of any type of data type

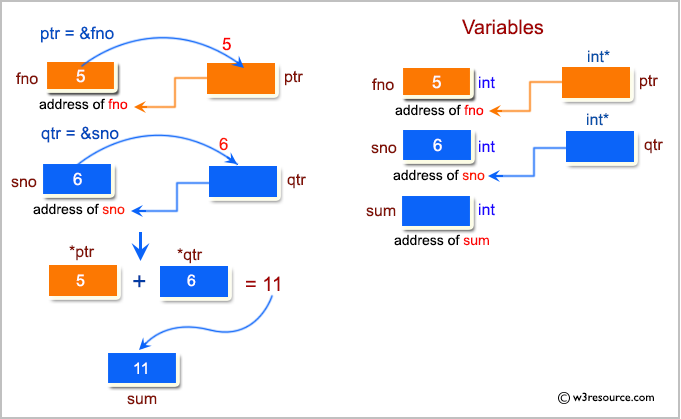
* 1. Void \*gptr ;
  2. int i = 10;
  3. char c = 'x';
  4. gptr = &i; (valid operation)
  5. gptr = &c; (valid operation)

While pointer variable is a variable which can hold address of a specific datatype for which you define type while declaration and it can hold address of that type only

1. int \*ptr ; (you declared this pointer variable as of type int)
2. int i = 10;
3. char c = 'x';
4. ptr = &i; (valid operation)
5. ptr = &c; (in valid operation)

25. Write a short note on pointers to pointers. Programming exercises (포인터에 대한 짧은 메모를 작성합니다. 프로그래밍 연습입니다.)





**C Code :**

#include <stdio.h>

int main()

{

int fno, sno, \*ptr, \*qtr, sum;

printf("\n\n Pointer : Add two numbers :\n");

printf("--------------------------------\n");

printf(" Input the first number : ");

scanf("%d", &fno);

printf(" Input the second number : ");

scanf("%d", &sno);

ptr = &fno;

qtr = &sno;

sum = \*ptr + \*qtr;

printf(" The sum of the entered numbers is : %d\n\n",sum);

return 0;

}

Sample Output:

Pointer : Add two numbers :

--------------------------------

Input the first number : 5

Input the second number : 6

The sum of the entered numbers is : 11

**Multiple-choice Questions**

1. The operator which compares two values is

(a) Assignment **(b) Relational** (c) Unary (d) Equality

2. Ternary operator operates on how many operands?

(a) 1 (b) 2 **(c) 3** (d) 4

3. Which operator produces the one’s complement of the given binary value?

(a) Logical AND (b) Bitwise AND (c) Logical OR **(d) bitwise NOT**

4. which operator has the lowest precedence?

(a) Sizeof (b) Unary (c) Assignment **(d) Comma**

5. Which of the following is the conversion character associated with short integer?

(a) %c **(b) %h** (c) %e (d) %f

6. Which of the following is not a character constant?

(a) ‘A’ **(b) “A”** (c) ‘ ‘ (d) ‘\*’

7. Which of the following is a valid variable name?

(a) Initial.Name (b) A+B (c) $amt **(d) Floats**

8. Which operator cannot be used with floating point numbers?

(a) + (b) – **(c) %** (d) \*

9. Identify the erroneous expression.

(a) X=y=2, 4; (b) res = ++a \* 5; **(c) res = /4;** (d) res = a++ -b \*2

10. Function declaration statement identifies a function with its

(a) Name (b) Arguments (c) Data type of return value **(d) All of these**

11. Which return type cannot return any value to the calling function?

(a) int (b) float **(c) void** (d) double

12. Memory is allocated for a function when the function is

(a) declared **(b) defined** (c) called (d) returned

13. \*($num) is equivalent to writing

(a) &num (b) \*num **(c) num** (d) None of these

14. Which operator retrieves the lvalue of a variable?

**(a) &** (b) \* (c) -> (d) None of these

15. Which operator is used to dereference a pointer?

(a) & **(b) \*** (c) -> (d) None of these

**True or False**

1. We can have only one function in a C program : False
2. Keywords are case sensitive. : True
3. Variable are case sensitive. : False
4. Signed variables can increase the maximum positive range. : False
5. Comment statements are not executed by the compiler. : True
6. Equality operators have higher precedence than the relational operators. : False

7. Shifting once to the left multiplies the number by 2. : True

8. Decision control statements are used to repeat the execution of a list of statements. : False

9. printf("%d", scanf("%d", &num)); is a valid C statement. : True

10. 1,234 is a valid integer constant. : False

11. A printf statement can generate only one line of output. : False

12. stdio.h is used to store the source code of the program. : False

13. The closing brace of main() is the logical end of the program. : True

14. The declaration section gives instructions to the computer. : True

15. Any valid printable ASCII character can be used for a variable name. False

16. Underscore can be used anywhere in the variable name. : False

17. void is a data type in C. : True

18. All arithmetic operators have same precedence. : False

19. The modulus operator can be used only with integers. : True

20. The calling function always passes parameters to the called function. : False

21. The name of a function is global. : True

22. No function can be declared within the body of another function. : True

23. The & operator retrieves the 1value of the variable. : True

24. Unary increment and decrement operators have greater precedence than the dereference operator. : True

25. On 32-bit systems, an integer variable is allocated 4 bytes. : True

**Fill in the blanks**

1. C was developed by \_\_\_\_\_. : Dennis Ritchie
2. The execution of a C program begins at \_\_\_\_. : main()
3. In the memory, characters are stored as \_\_\_\_\_. : ASCII codes
4. Return 0 rerurns 0 to the \_\_\_\_\_\_\_. : Operating system
5. \_\_\_\_\_\_\_ finds the remainder of an integer division. : Modulus operator(%)
6. Sizeof is a \_\_\_\_\_ operator used to calculate the sizes of data types. : Unary
7. \_\_\_\_\_\_\_ is also knows as forced conversion. : Typecasting
8. \_\_\_\_\_\_\_ is executed when the value of the variable does not match with any of the values of the case statement. : Default case
9. \_\_\_\_\_\_\_ function prints data on the monitor. : printf()
10. A C program ends with a \_\_\_\_\_\_. : Closing bracket
11. \_\_\_\_\_ causes the cursor to move to the next line. : \n
12. A variable can be made constant by declaring it with the qualifier \_\_\_\_\_ at the time of initializa-tion. : const
13. \_\_\_\_\_ operator returns the number of bytes occupied by the operand. : sizeof
14. The \_\_\_\_\_ specification is used to read/write a short integer. : %hd
15. The \_\_\_\_\_ specification is used to read/write a hexadecimal integer. : %x
16. To print the data left-justified, \_\_\_\_\_ specif-ication is used. : -
17. After the function is executed, the control passes back to the \_\_\_\_\_. : Calling function
18. A function that uses another function is known as the \_\_\_\_\_. : Calling function
19. The inputs that the function takes are known as \_\_\_\_\_. : Arguments/parameters
20. Function definition consist of \_\_\_\_\_ and \_\_\_\_\_\_. : Function header and function body
21. In \_\_\_\_\_ method, address of the variable is passed by the calling function to the called function. : Call by reference
22. Size of character pointer is \_\_\_\_\_. : I byte
23. \_\_\_\_\_ pointer does not point to any valid memory address. : NULL
24. The \_\_\_\_\_ appears on the right side of the assignment statement. : Rvalue
25. The \_\_\_\_\_ operator informs the compiler that the variable is a pointer variable. : \*