Basic Information

1. we cant assign a declaration to a variable. Ex: int x=int a; int x=void fun();

2. Nested comments can't be used by c language. /\* // \*/ /\* /\* \*/ \*/

3. every executable and declarative statement must be terminated by semicolon. Semicolon represents the no of statements.

4. By using :: (scope resolution operator) we can access only global variables.

5. We must declare the variable before using it. Ex: x=10; //This is invalid int x;

 6. we cant apply = operator to the constant variables.

 7. All local variables are created whenever execution control entered inside the block and initialized by garbage value.+

Practice Sheet

**1. Identify the correct expression.**

**(a)a=b=3=4;            (b)a=b=c=d=0;**

**(c)float a=int b= 3.5; (d)int a; float b; a=b=3.5;**

**2. Which of the following about the C comments is incorrect ?**

**(a) Comments can go over multiple lines**

**(b) Comments can start anywhere in the line**

**(c) A line can contain comments without any language statements**

**(d) Comments can occur within comments**

**3. The C language terminator is**

**(a) semicolon   (b) colon     (c) period    (d) exclamation mark**

**4. void main(){**

**int a=10;**

**{**

**int a=20;**

**printf("%d %d",a,::a);**

**}**

**}**

**5. void main(){**

**printf("%d",a);**

**{**

**int a=10;**

**}**

**}**

**6. void main(){**

**char ch=65;**

**printf("%d",ch==A);**

**}**

**7. int x=1;**

**void main()**

**{**

**int x=10;**

**{**

**int x=20;**

**printf("%d %d ",x, ::x);**

**}**

**printf("%d  %d ",x,::x);**

**}**

**8. int x=10;**

**void main()**

**{**

**int x=1;**

**printf("%d %d",x,::x);**

**}**

**9. Which is logical error**

**a. Missing ;**

**b. Definition of function/variable not found**

**c. Declaration of function/variable not found**

**d. Infinite loop**

**10.Which type of errors don’t terminate execution of program**

**a. Syntax errors          b. Linker errors**

**c. Runtime errors         d. Logical errors**

**11.Most dangerous errors are :**

**a. Compile time            b. Runtime**

**c. Logical                d. Linker error**

**12.\_\_\_\_\_ translates low level language to machine lang.**

**a. Assembler              b. Compiler**

**c. Decompiler             d. Interpreter**

**13. //identify error**

**void main()**

**{**

**char x,y;**

**int z;**

**x=a;**

**y=b;**

**z=x+y;**

**printf("%d",z);**

**getch();**

**}**

**14.void main()**

**{**

**const char x='Z';**

**int a=4;**

**a=5;**

**printf(" %d",a);**

**x='A';**

**printf(" %c",x);**

**getch();**

**}**

Basic Concept

**There are various c compilers are variables. Some of these are:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.N.** | **Name** | **Microprocessor** | **OS** |
| **1** | Turbo c 3.0 | 8086 | MS DOS |
| **2** | ANSIC C | 80386 | LINUX |
| **3** | Borland C 4.0 | 80386 | WINDOW |
| **4** | Microsoft C | 8086 | MS DOS |
| **5** | Visual C++ | 80386 | WINDOW |

Note: 8086 is 16 bit microprocessor while 80386 is 32 bit microprocessor.

Note: Different versions of compilers are based on the different microprocessors and support many OS. It is always changing. As a programmer you should know the microprocessor name, its world length etc. which your compiler is based on. Since c language is platform dependent. In preprocessor section you will know how to make a program as much as platform independent.

**Turbo c compiler :**

Turbo c is an IDE of c programming language created by Borland. Turbo C 3.0  is based on MS DOS operation  system. It is one of the most popular c compilers. It uses 8086 microprocessor which is 16 bit microprocessor. It has 20 address buses and 16 data bus. Its word length is two byte.   
Size of data types in Turbo C 3.0:

|  |  |
| --- | --- |
| **Data type** | **Size** |
| short int | 2 |
| Int | 2 |
| long int | 4 |
| Char | 1 |
| Float | 4 |
| Double | 8 |
| long double | 10 |

Byte ordering : Little Endianness  
Default pointer : Near  
Default memory model : Small  
To compile a c program: Alt + F9  
To run a c program: Ctrl + F9

Turbo C 4.5 is based on Microsoft window operating system. It is 32 bit compilers.

Size of data type in Turbo C 4.5:

|  |  |
| --- | --- |
| **Data type** | **Size (Byte)** |
| short int | 2 |
| Int | 4 |
| long int | 4 |
| Char | 1 |
| Float | 4 |
| Double | 10 |
| long double | 12 |

Default pointer: Far

Default memory model: Compact

**Hexadecimal representation in c :**

In hexadecimal number system we use 16 different digits so its base is 16. List of all hexadecimal digits:

|  |  |  |
| --- | --- | --- |
| **Hexadecimal digit** | **Decimal equivalent** | **Binary equivalent** |
| **0** | 0 | 0000 |
| **1** | 1 | 0001 |
| **2** | 2 | 0010 |
| **3** | 3 | 0011 |
| **4** | 4 | 0100 |
| **5** | 5 | 0101 |
| **6** | 6 | 0110 |
| **7** | 7 | 0111 |
| **8** | 8 | 1000 |
| **9** | 9 | 1001 |
| **A** | 10 | 1010 |
| **B** | 11 | 1011 |
| **C** | 12 | 1100 |
| **D** | 13 | 1101 |
| **E** | 14 | 1110 |
| **F** | 15 | 1111 |

To convert the binary number into hexadecimal number:

Make the group of four binary digits from right to left and replace the four binary digits with the equivalent hexadecimal digit using above table.

For example:

Binary number = 11000111110101

Group of four digits from right side:



to make group of four digit of left most  digit 11 , add two zero at the left side i.e. 0011

Now put or replace with it the equivalent hexadecimal digit using above table



So, equivalent hexadecimal number will be 31F5

**What will be address range which can be represented in 20 bit?**

|  |  |  |
| --- | --- | --- |
|  | **Binary** | **Hexadecimal** |
| **Min possible** | 0000 0000 0000 0000 0000 | 00000 |
| **Max possible** | 1111 1111 1111 1111 1111 | FFFFF |

In c any hexadecimal number start with 0x 0r 0X So, address range   will be 0x00000 to 0xFFFFF. So in turbo C 3.0 memory address of all variables must be within 0x00000 to oxFFFFF.

It is 1MB memory range.

Note.

2^10 = 1KB

2^20 = 1MB

2^30 = 1GB

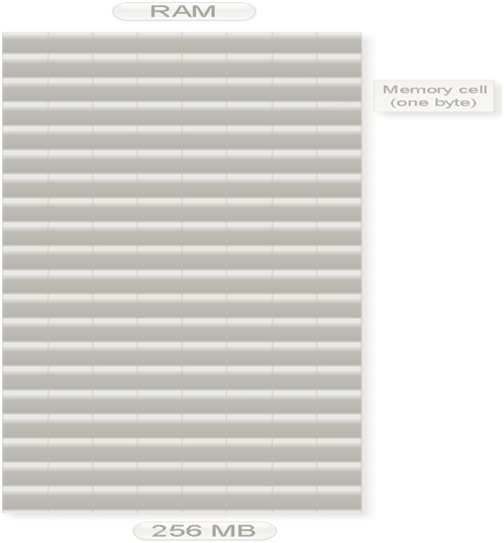
Where 10, 20, 30 are number of bit

**What is the ram memory: Memory cell in computer**

Entire RAM has divided in numbers of equal parts, which are

known as memory cells. Following diagram represents the 256

MB RAM.



Each cell can store one-byte data. Data are stored in the   
binary number system. That is a character data reserves one memory cell while floating data reserves four memory cells.

Each memory cell has unique address. Address is always in  
whole number and must be in increasing order. We will discuss  
how a characters, integers etc. data are in stored in the   
data type chapter. Just for now assume

**int** a = 4;

Here variables a stores in the memory in the flowing way:



If you know memory address of first cell is 0x5000 then   
what would be the memory address of next memory cell?

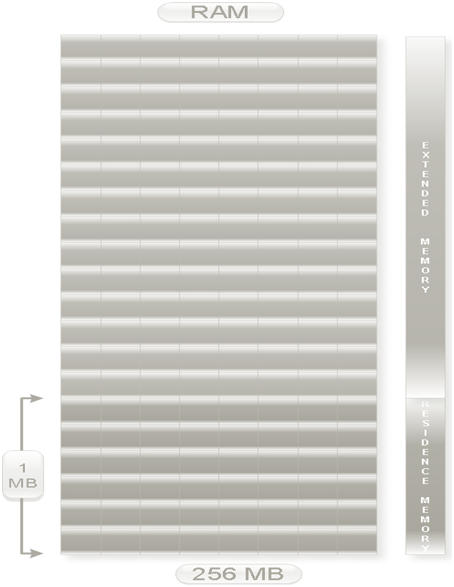
It will 5001 since integer data always stores at   
continuous memory location and as we know memory address   
always in increasing order.

**What is resident memory of a computer?**

RAM has divided into two parts:

(1)        Extended memory (useless)

(2)        Residence memory



In Turbo C 3.0 compiler size of residence memory is 1MB.

**resident memory:**

When any program is executed it is stored in the   
residence memory. For turbo c 3.0, it has 1MB residence   
memory i.e. when we open turbo c 3.0 it stores 1MB in the   
RAM.

**Offset address and segment number in c programming language**

Each segment has divided into two parts.

1. Segment no (4 bit)

2. Offset address (16 bit)



So, in the other words we can say memory address of any variable in c has two parts segment number and offset address.

In turbo c 3.0 a particular segment number offset address varies from 0x0000 to 0xFFFF

Suppose physical address of any variable in c is 0x500F1.

Then its segment number is 5 and offset address is 00F1.

Write a program to find the offset address of any variable?

#include<stdio.h>

int main(){

int x;

printf("%u ",&x); //To print offset address

printf("%p ",x); //To print segment address

printf("%p ",&x); //To print offset address

printf("%fp ",&x); //To print segment address : offset address

return 0;

}