🎷

Saxxy Things

- If you write-

  ```
  int i=10;
  repeat(i=i-1){
    cout<<".";
  } //This will print 9 '.' (dots)
  ```

  The reason for this is that the *standard assignment operator returns the value of the right expression* so we are essentially writing --i where we have written i=i-1 in the repeat statement.

- %, * and / all have the same precedence and +, - have equal precedence lower than that of the first 3. You look from left to right for operators of same preference.

- There is a precedence in logical operators (!,&&, ||)-

  ```
  #include<simplecpp>

  main_program{ //BTW NOT (!) has the highest precedence followed by && and then ||
      int x=10, y=10, z=9;
      if(x==5&&y==10||z==9)
          cout<<"iCareee"<<endl; //Gets printed
      else
          cout<<"Bubye";
      if(x==5&&++y==11) //Since first statement is false, the second condition isn't even checked hence y doesn't increment
          cout<<"nutin"; //doesn't get printed
      if(x==10||z++==9) //Since first statement is true, we don't check second statement thus z is still 9
          cout<<"Nutin again"<<endl; //gets printed
      cout<<y<<z; //prints '109'
      // !y==10||z>=12&&x!=1 is same as (!(y==10))||(z>=12&&x!=1)
  ```

- You can use scope of a variable even with shapes in Canvas

  ```
      {
          Text t(100, 100, "Yess"); //This prints Yess at (100, 100)
          getClick();
      } //t now gets destroyed
      Text t(100, 100, "Nooo"); //This new t now does the work
      getClick();
  ```

- Assignment operator might be used as a condition instead of the equality wala relational operator. Basically if I had written, i=0, it would have returned 0 to ‘if’ and then block would not have been executed while for any i≠0, it would have returned true.

```
int i=0;
if(i=10){
  cout<<"In case you haven't noticed it yet, it reads i=10 instead of i==10."<<endl;
  cout<<"What happens is that assignment operation returns the RHS for some reason ";
}
if(i=0){
  cout<<"i=0 makes the condition statement false. Had it been i=7 or 9 or basically any non zero value, it would have returned true";
}

cout<<i;
/*Output-
In case you haven't noticed it yet, it reads i=10 instead of i==10.
What happens is that assignment operation returns RHS for some reason 0
*/
```

- You can make life easier using #define line

```
#include<simplecpp>

#define ll long long int //You can use this to streamline your typing
#define uwuIsThatAnInteger int

main_program{
  ll yea=0;
  uwuIsThatAnInteger yes_it_is=1e5;
  cout<<yes_it_is<<yea;
}
```

- You can make object arrays with Canvas

```
Circle c[20];
for(int i=0;i<=9;i++){
  c[i]=Circle(250, 250, i);
}
```

- Local variables are given higher preference than the variables with larger scope

```
int x=10;
{
  int x=9;
  {
    int x=8;
    cout<<x;
  }
  cout<<"\n"<<x;
}
cout<<endl<<x;
/*Not only does this have no syntax errors, it also gives the following output
8
9
10
*/ //Note- There is no way to access the other x's in the innermost block due to shadowing
//however for global variables you can use ::x
```

- The * thingy associates to the immediate right

```
int* v, p;
//Here p is an int and v is a int pointer i.e. *v is int
int *w;
double *k;
v=&p;
w=v;
p=10;
*v++;
cout<<p<<' '<<*w; //11 11
//k=w; //This last line will give an error
```

- The “&” can be used for 2 different things- pass by reference and extracting the memory address of a specific datatype. “*” can be used for dereferencing as well as defining pointers

  ```
  void swapper(int &a, int &b){ //This is pass by reference and not passing pointers
    int m=a;
    a=b;
    b=m;
    return;
  }

  int lol(int* &m){
    //We just got the address of c via pass by reference
    return m+1;
  }

  main_program{
    int a,b,c, *d;
    cin>>a>>b>>c;
    d=&c; //d becomes the pointer of c
    swap(a,b); //swaps a with b
    swap(b,*d); //swaps b with c
    lol(d); //Even if there isn't anything to recieve the return, it doesn't matter
   //Thus now original value of a is in c, that of b is in a and that of c is in b
  }
  ```

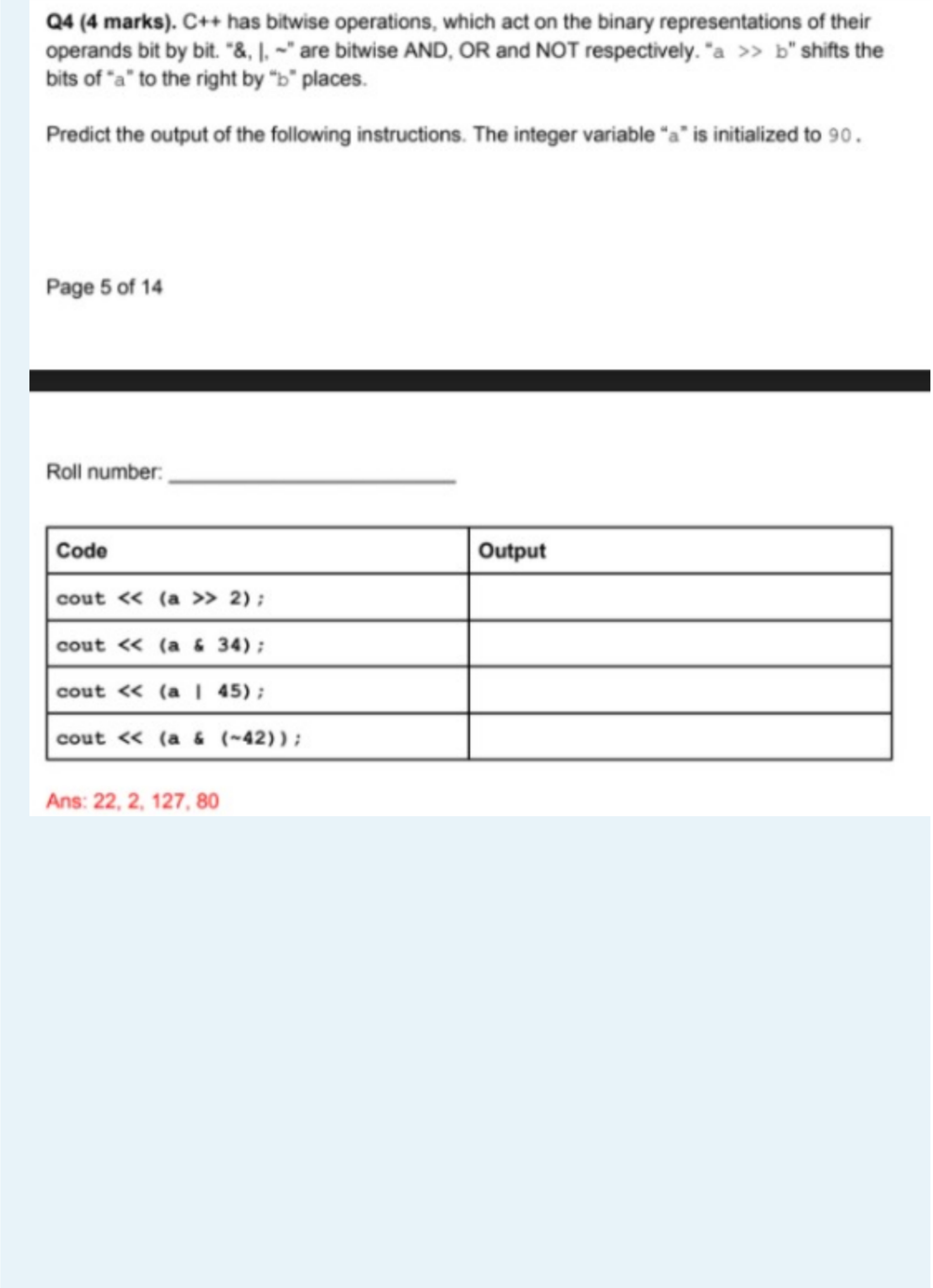
- Precision wala thing is interesting
  There are various format flags when it comes to precision. “Floatfield Format Flag” decides the format in which floats are represented. It has multiple flag values- fixed (where precision decides no of significant digits after decimal point), scientifc (where float is represented in scientific fashion and precision decides significant digits after decimal) and none/default (where **precision decides total number** of signifcant digits and it can use either notation to achieve that)

  ```
  cout<<precision(2);
  cout<<123<<"\t"<<120.23<<"\t"<<120.0<<"\t"<<3.472;
  //prints 123 1.2e+002  1.2e+002   3.5
  //Cause 123 is an integer so we couldn't give any fucks about it. Also note how it went to scientific just to maintain precision
  cout<<fixed;
  cout.precision(7);
  cout<<12<<"\t"<<12.0<<"\t"<<12.52431242<<"\t"<<endl;
  //Prints 12    12.0000000    12.5243124
  cout<<scientific;
  cout.precision(2);
  cout<<123<<"\t"<<123.4<<"\t"<<3.223<<endl;
  //prints 123    1.23e+002   3.22e+000
  ```

- Hex code- The prefix ‘0x’ tells the compiler ki iske aage mera hex code hai

  ```
  int x=0x3, y=0xaa, z=0x10; //0xaa is the same as 0xAA
  cout<<x<<"\t"<<y<<"\t"<<z;
  //prints 3      170       16
  ```

- Bitwise operators ‘&, ~ ,| ,>> exist”. An easy example to understand them



Q4 (4 marks). C++ has bitwise operations, which act on the binary representations of their operands bit by bit. "&, |, ~" are bitwise AND, OR and NOT respectively. "`a >> b`" shifts the bits of "`a`" to the right by "`b`" places.

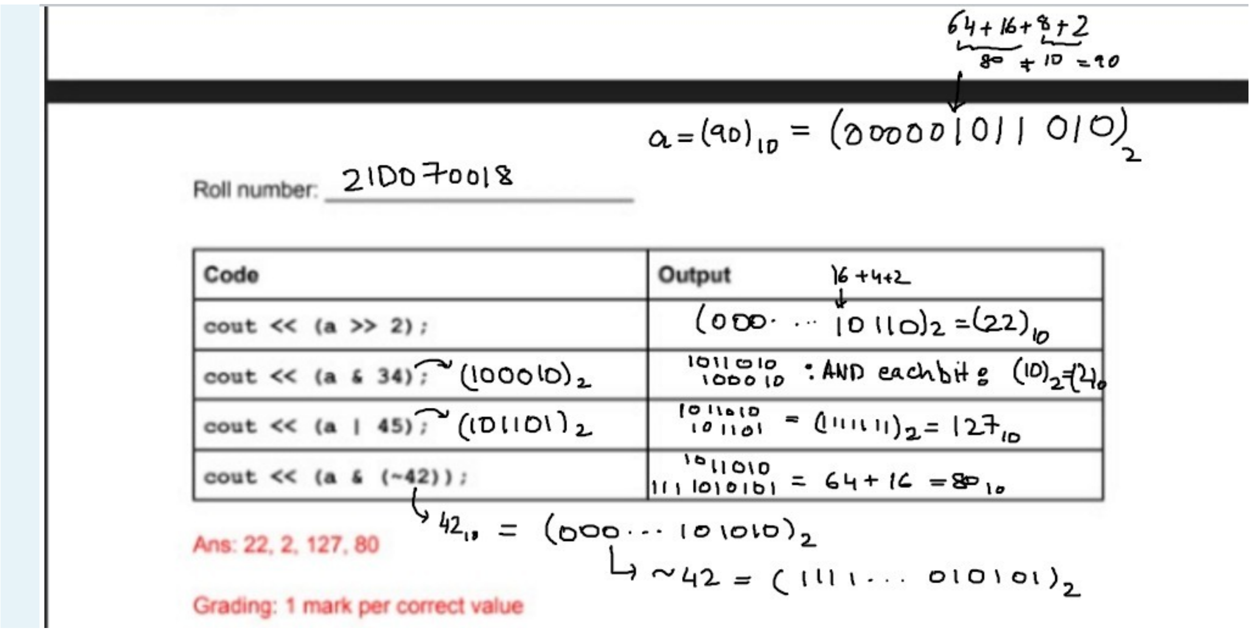
Predict the output of the following instructions. The integer variable "`a`" is initialized to `90`.

Page 5 of 14

Roll number: ______________________

Code	**Output**
`cout << (a >> 2);`	
`cout << (a & 34);`	
`cout << (a | 45);`	
`cout << (a & (~42));`	

Ans: 22, 2, 127, 80



$\underbrace{64+16}_{80}+\underbrace{8+2}_{10}=90$

$a=(90)_{10}=(00000\,1011\,010)_2$

Roll number: 210070018

Code	Output $16+4+2$
`cout << (a >> 2);`	$(000\cdots10110)_2=(22)_{10}$
`cout << (a & 34);` $(100010)_2$	$\begin{matrix}1011010\\100010\end{matrix}$: AND eachbit : $(10)_2=(2)_{10}$
`cout << (a | 45);` $(101101)_2$	$\begin{matrix}1011010\\101101\end{matrix}=(1111111)_2=127_{10}$
`cout << (a & (~42));`	$\begin{matrix}1011010\\111\,1010101\end{matrix}=64+16=80_{10}$

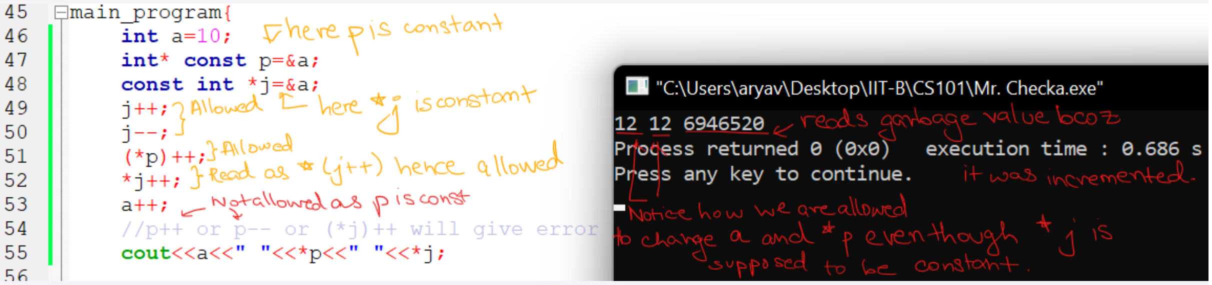
$42_{10}=(000\cdots101010)_2$

$\hookrightarrow\ \sim42=(1111\cdots010101)_2$

Ans: 22, 2, 127, 80

Grading: 1 mark per correct value

- const int *p and int* const p are not the same



```
main_program{
    int a=10;
    int* const p=&a;
    const int *j=&a;
    j++;
    j--;
    (*p)++;
    *j++;
    a++;
    //p++ or p-- or (*j)++ will give error
    cout<<a<<" "<<*p<<" "<<*j;
```

here p is constant

Allowed ← here *j is constant

Allowed

Read as *(j++) hence allowed

← Not allowed as p is const

"C:\Users\aryav\Desktop\IIT-B\CS101\Mr. Checka.exe"

```
12 12 6946520
Process returned 0 (0x0)   execution time : 0.686 s
Press any key to continue.
```

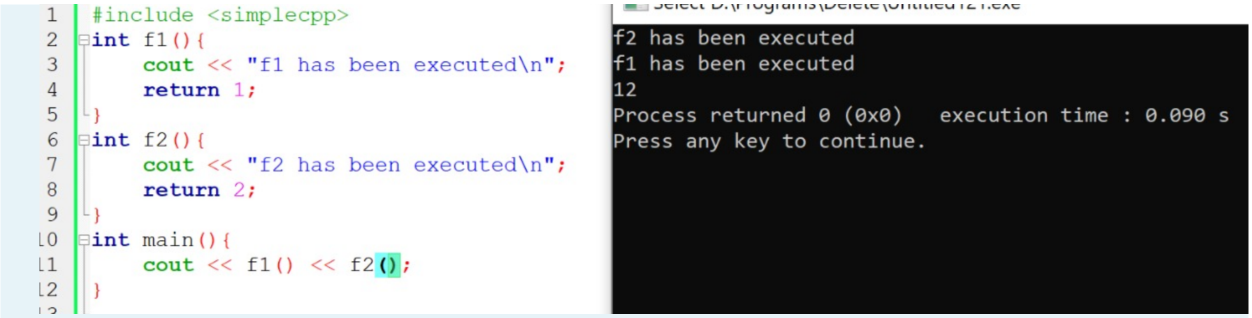
← reads garbage value bcoz it was incremented.

Notice how we are allowed to change a and *p even though *j is supposed to be constant.

const takes the things to its right as constant always. Although, if you are talking about const int a and int const a, they would be the same. Just to cement the point written in the bottom right with red-

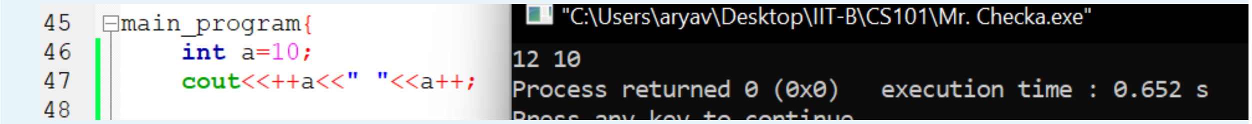
```
int b=100, c=200, d=150;
const int *m=&b; //You cannot change the value of *m but m can be changed
int const *l=&c; //Since *l is there on the right of const, same as above
int* const n=&d; //Here n is const but you can change *n
// *n++ is invalid as it is read as *(n++)
(*n)++; //Valid. So is *n+=1;
//(*m)++; This is invalid but the next line isn't
b++; //Cause b don't give no fucks about what *m thinks
```

- A sexy thing cout does- It prints stuff from left to right but it executes the values from right to left.



```
#include <simplecpp>
int f1(){
    cout << "f1 has been executed\n";
    return 1;
}
int f2(){
    cout << "f2 has been executed\n";
    return 2;
}
int main(){
    cout << f1() << f2();
}
```

```
f2 has been executed
f1 has been executed
12
Process returned 0 (0x0)   execution time : 0.090 s
Press any key to continue.
```

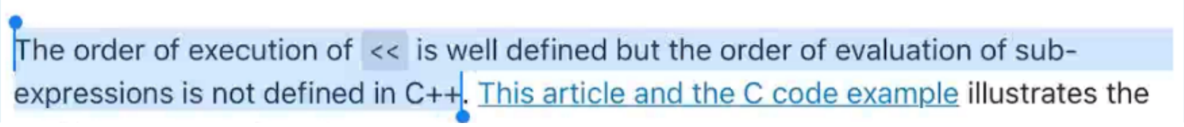


```
main_program{
    int a=10;
    cout<<++a<<" "<<a++;
```

"C:\Users\aryav\Desktop\IIT-B\CS101\Mr. Checka.exe"

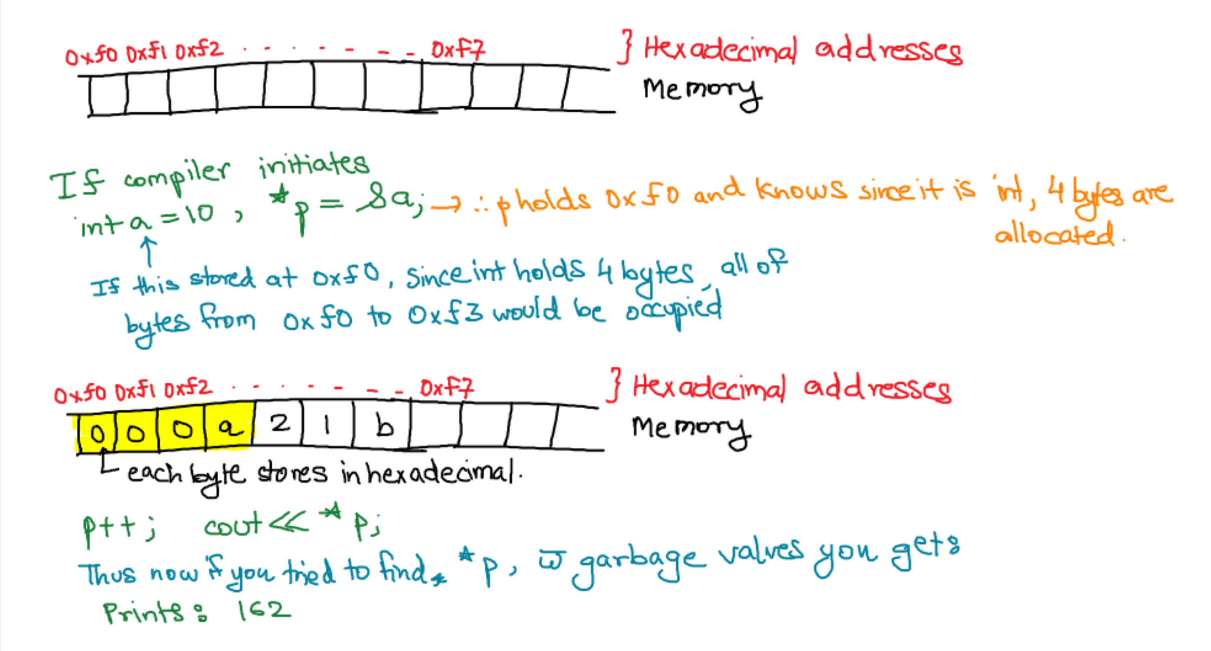
```
12 10
Process returned 0 (0x0)   execution time : 0.652 s
Press any key to continue
```

(This order is only for our GCC compiler. Otherwise C++ in general does not have any guarantee ki kaunsa pehle hoga execute)



The order of execution of `<<` is well defined but the order of evaluation of sub-expressions is not defined in C++. This article and the C code example illustrates the

- Now an irrelevant concept on addresses and pointers



0xf0 0xf1 0xf2 - - - - - - - - 0xf7
} Hexadecimal addresses
Memory

If compiler initiates int a = 10 , *p = &a; → ∴ p holds 0xf0 and knows since it is 'int', 4 bytes are allocated.

↑ If this stored at 0xf0, since int holds 4 bytes, all of bytes from 0xf0 to 0xf3 would be occupied

0xf0 0xf1 0xf2 - - - - - - - - 0xf7
} Hexadecimal addresses
0
0
0
a
2
1
b
Memory

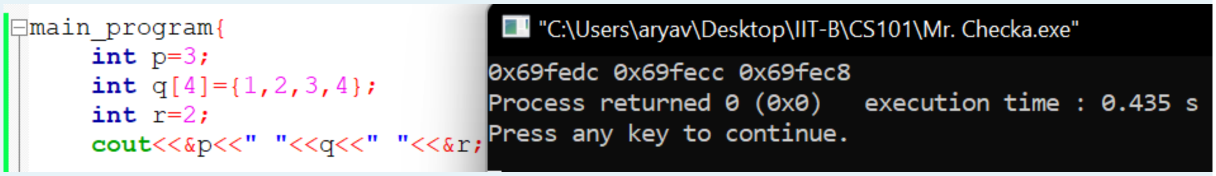
each byte stores in hexadecimal.

p++; cout<< *p;

Thus now if you tried to find *p, ϖ garbage values you gets:

Prints: 162

- Memory allocation does some sussy baka stuff



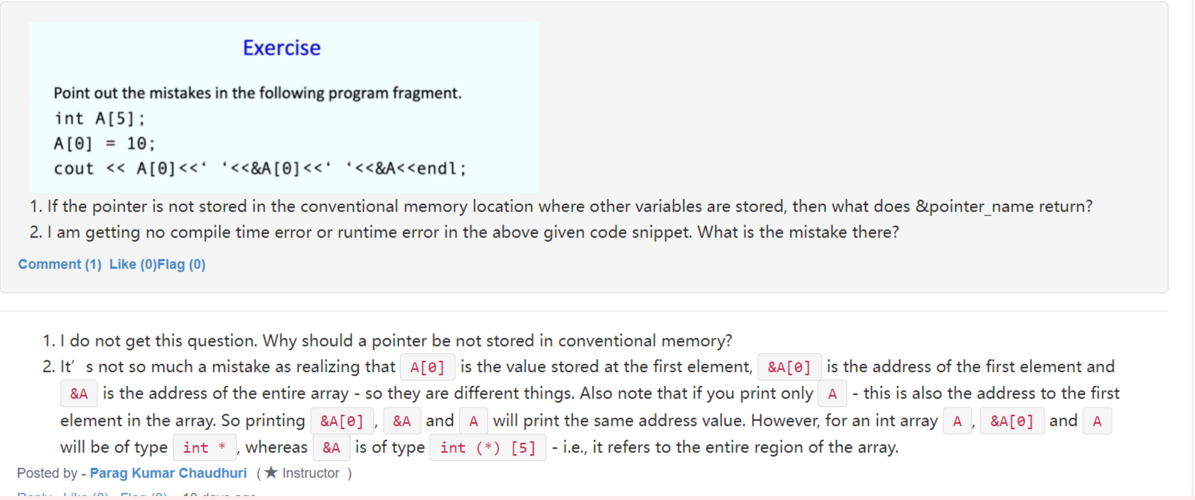
```
main_program{
    int p=3;
    int q[4]={1,2,3,4};
    int r=2;
    cout<<&p<<" "<<q<<" "<<&r;
```

"C:\Users\aryav\Desktop\IIT-B\CS101\Mr. Checka.exe"

```
0x69fedc 0x69fecc 0x69fec8
Process returned 0 (0x0)   execution time : 0.435 s
Press any key to continue.
```

You can clearly see that p, q, r have descending instead of ascending memory addresses (This **does not** mean that we store information is a descending manner)

- 2 good questions-



Exercise

Point out the mistakes in the following program fragment.

```
int A[5];
A[0] = 10;
cout << A[0]<<‘ ‘<<&A[0]<<‘ ‘<<&A<<endl;
```

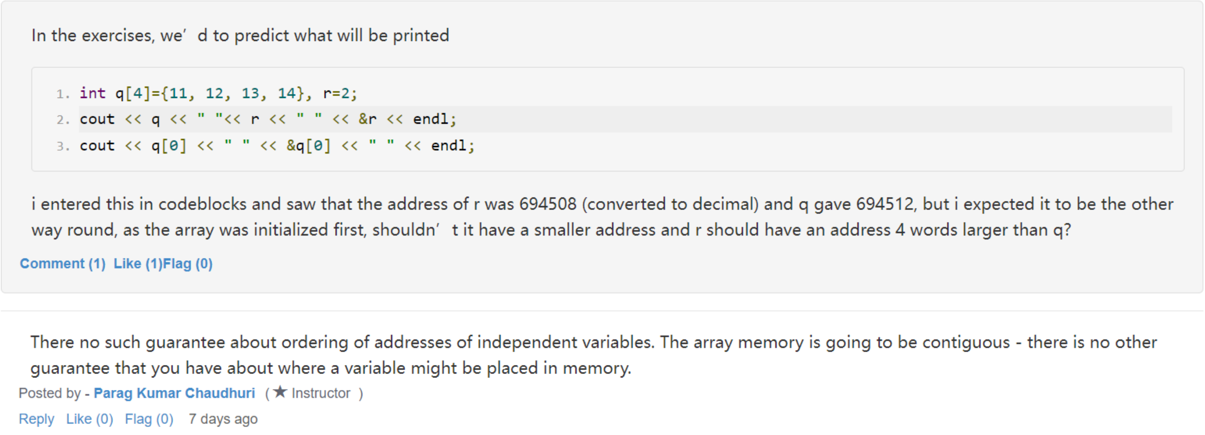
1. If the pointer is not stored in the conventional memory location where other variables are stored, then what does &pointer_name return?
2. I am getting no compile time error or runtime error in the above given code snippet. What is the mistake there?

Comment (1) Like (0)Flag (0)

1. I do not get this question. Why should a pointer be not stored in conventional memory?
2. It’ s not so much a mistake as realizing that `A[0]` is the value stored at the first element, `&A[0]` is the address of the first element and `&A` is the address of the entire array - so they are different things. Also note that if you print only `A` - this is also the address to the first element in the array. So printing `&A[0]` , `&A` and `A` will print the same address value. However, for an int array `A` , `&A[0]` and `A` will be of type `int *` , whereas `&A` is of type `int (*) [5]` - i.e., it refers to the entire region of the array.

Posted by - Parag Kumar Chaudhuri (★ Instructor)

Reply Like (0) Flag (0) 10 days ago



In the exercises, we’ d to predict what will be printed

```
int q[4]={11, 12, 13, 14}, r=2;
cout << q << " "<< r << " " << &r << endl;
cout << q[0] << " " << &q[0] << " " << endl;
```

i entered this in codeblocks and saw that the address of r was 694508 (converted to decimal) and q gave 694512, but i expected it to be the other way round, as the array was initialized first, shouldn’ t it have a smaller address and r should have an address 4 words larger than q?

Comment (1) Like (1)Flag (0)

There no such guarantee about ordering of addresses of independent variables. The array memory is going to be contiguous - there is no other guarantee that you have about where a variable might be placed in memory.

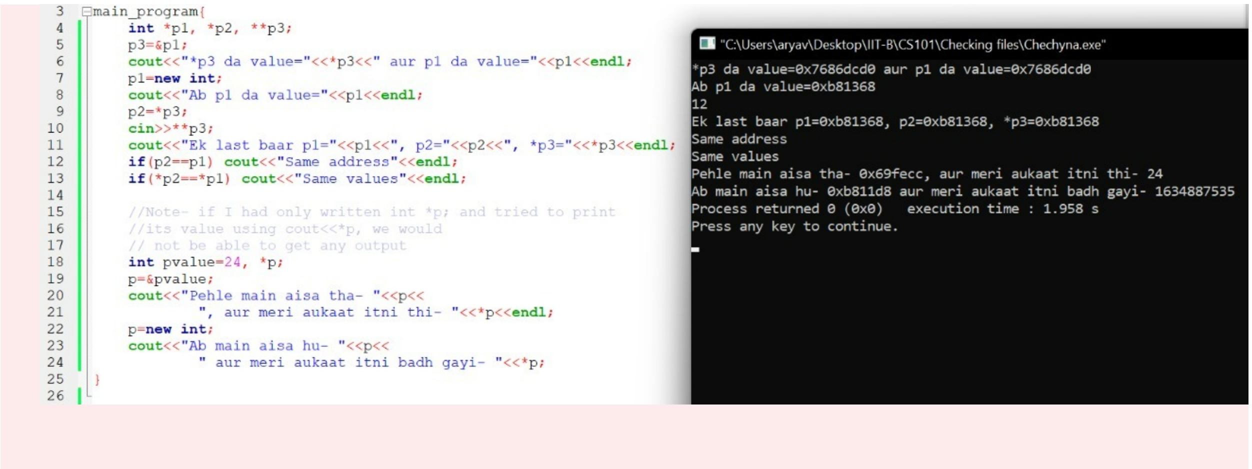
Posted by - Parag Kumar Chaudhuri (★ Instructor)

Reply Like (0) Flag (0) 7 days ago

- You can “format” a pointer i.e. make it stop pointing in any direction by this

```
...
int *p=&some_integer;
p=NULL;
```

- Dekho inhe (not really in portion but it is related to heap and stack)



```
main_program{
    int *p1, *p2, **p3;
    p3=&p1;
    cout<<"*p3 da value="<<*p3<<" aur p1 da value="<<p1<<endl;
    p1=new int;
    cout<<"Ab p1 da value="<<p1<<endl;
    p2=*p3;
    cin>>**p3;
    cout<<"Ek last baar p1="<<p1<<", p2="<<p2<<", *p3="<<*p3<<endl;
    if(p2==p1) cout<<"Same address"<<endl;
    if(*p2==*p1) cout<<"Same values"<<endl;

    //Note- if I had only written int *p; and tried to print
    //its value using cout<<*p, we would
    // not be able to get any output
    int pvalue=24, *p;
    p=&pvalue;
    cout<<"Pehle main aisa tha- "<<p<<
            ", aur meri aukaat itni thi- "<<*p<<endl;
    p=new int;
    cout<<"Ab main aisa hu- "<<p<<
            " aur meri aukaat itni badh gayi- "<<*p;
}
```

"C:\Users\aryav\Desktop\IIT-B\CS101\Checking files\Chechyna.exe"

```
*p3 da value=0x7686dcd0 aur p1 da value=0x7686dcd0
Ab p1 da value=0xb81368
12
Ek last baar p1=0xb81368, p2=0xb81368, *p3=0xb81368
Same address
Same values
Pehle main aisa tha- 0x69fecc, aur meri aukaat itni thi- 24
Ab main aisa hu- 0xb811d8 aur meri aukaat itni badh gayi- 1634887535
Process returned 0 (0x0)   execution time : 1.958 s
Press any key to continue.
```