

Dynamic Memory Allocation & Typecasting | C++...

And Structures, Unions, and Enumerations | C++

Dynamic Memory Allocation

DMA in C++ refers to performing memory allocation manually by programmer.

Dynamically allocated memory is allocated on **Heap**.

Applications:

- Variable length arrays
- Linked lists, Trees, Graphs, etc.

For normal variables, memory is allocated and deallocated automatically.

For dynamically allocated memory, it is our responsibility to deallocate memory when no longer needed.

int a; *(Handwritten note: DMA)*
malloc
free
new
delete

Dynamic Memory Allocation

Dynamic Memory Allocation in C is using malloc(), calloc(), free() and realloc().

C++ supports these functions and has two operators **new** and **delete** that perform the task of allocating and freeing the memory in a easier way!

Operator new acquires its memory by calling an operator **new()**. Operator delete frees its memory by calling an operator **delete()**.

unacademy

new Operator

new operator requests for memory allocation on the **Free Store**.

If sufficient memory is available, **new** operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer.

new char;
ptr = new data_type;

eg:
`(int *ptr) = NULL;`
`ptr = new int;`

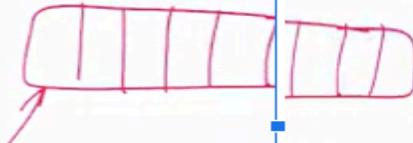
100 chars 1B
4B

new Operator

when new needs memory on the free store for an object of type X, it calls operator:

new(sizeof(X))

int



object of type X, it calls

Similarly, when new needs memory on the free store for an array of N objects of type X, it calls operator:

new[](N*sizeof(X))

int *p = new int[10]



9:10 / 46:12



delete Operator

It is our responsibility to deallocate dynamically allocated memory!

If you forget to delete the allocated memory, memory leak happens.

`delete ptr;`

`delete[] ptr;`



Practice Questions

- How to create a dynamic array of integers of size 3 using new in C++?

A `int *arr = new int*[3];`

int* ptr = new int



B `int **arr = new int *[3];`

C `int arr = new int [3];`

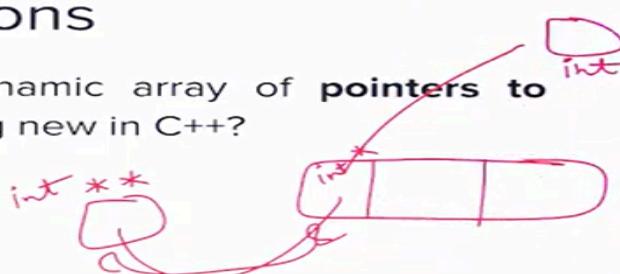
D `int *arr = new int [3];`



Practice Questions

- How to create a dynamic array of **pointers to** integers of size 3 using new in C++?

A int *arr = new int *[3];
B int **arr = new int *[3];
C int arr = new int [3];
D int *arr = new int [3];



Practice Questions

- Correct statement(s)?

A new is an operator
B malloc() is a function
C new allocates memory from free store
D malloc() allocates memory from heap
E None of the above

Example

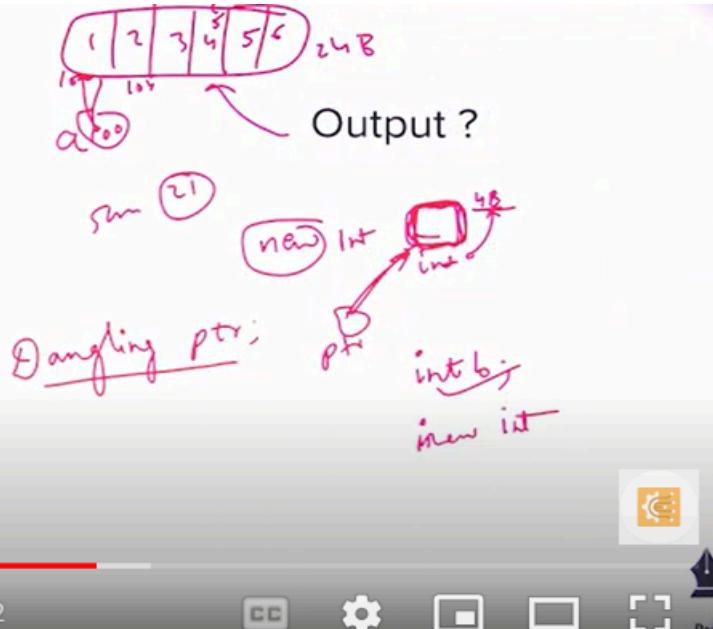
```
#include<iostream>
using namespace std;
int main(){
    int *a, sum = 0;
    a = new int[6];
    for(int i=0; i<6; i++){
        a[i] = i+1;
    }
    for(int i=0; i<6; i++){
        sum+=a[i];
    }
    for(int i=0; i<6; i++){
        sum+=a[i];
    }
    cout<<sum<<endl;
    return 0;
}
```

Output

42

Example

```
#include<iostream>
using namespace std;
int main(){
    int *a, sum = 0;
    a = new int[6];
    for(int i=0; i<6; i++){
        a[i] = i+1;
    }
    for(int i=0; i<6; i++){
        sum+=a[i];
    }
    delete[] a;
    for(int i=0; i<6; i++){
        sum+=a[i];
    }
    cout<<sum<<endl;
    return 0;
}
```



Output ?

Not known
Dangling Pointe

Example

```
#include<iostream>
using namespace std;
int main(){
    int *a, sum = 0;
    a = new int[6];
    for(int i=0; i<6; i++){
        a[i] = i+1;
    }
    for(int i=0; i<6; i++){
        sum+=a[i];
    }
    delete[] a;
    a = NULL;
    for(int i=0; i<6; i++){
        sum+=a[i];
    }
    cout<<sum<<endl;
    return 0;
}
```

Output ?

Segmentation fault

.....

Typecasting

Typecasting

Typecasting is making a variable of one type, such as an int, act like another type, a char, for one single operation.

To typecast something, put the type of variable you want the actual variable to act as inside parentheses in front of the actual variable.

```
#include <iostream>
using namespace std;
int main() {
    cout << (char)66 << endl;
}
```

Implicit type conversion in C++

C++ supports other types to typecast as well.

bool -> char -> short int -> int -> unsigned int -> long -> unsigned -> long long
-> float -> double -> long double

Data loss can occur (Signs, overflow)

Practice Questions

Which of these is/are a valid typecast in C++?

- A. a(char);
- B. char:a;
- C. (char)a; ✓
- D. to(char, a);
- E. char(a); ✓

Example

#include <iostream>
using namespace std;
int main(){
 int a = 257;
 char ch = (char)a + 'a';
 printf("%c\n", ch);
 return 0;
}

The diagram illustrates the memory representation and output of the above C program. It shows two memory structures: one for variables and one for the stack.

- Memory Structure:** Shows four bytes (LSB at the bottom) with values 101, 102, 103, 104. Red annotations indicate 'little endians' and 'big endians'.
- Stack:** Shows the variable 'a' (size 4) and the character 'ch' (size 1). The value of 'a' is shown as 10000001 (big-endian) and 101 102 103 104 (little-endian).
- Output:** The output is circled in red and labeled 'Output?'. It shows the character 'c' (ASCII 99), which corresponds to the value 101 102 103 104 (little-endian) in memory.
- Annotations:** Handwritten notes include 'little', 'big', 'LSB', 'MSB', 'a', 'ch', 'printf("%c\n", ch)', and 'return 0;'. A large red circle highlights the output character 'c'.

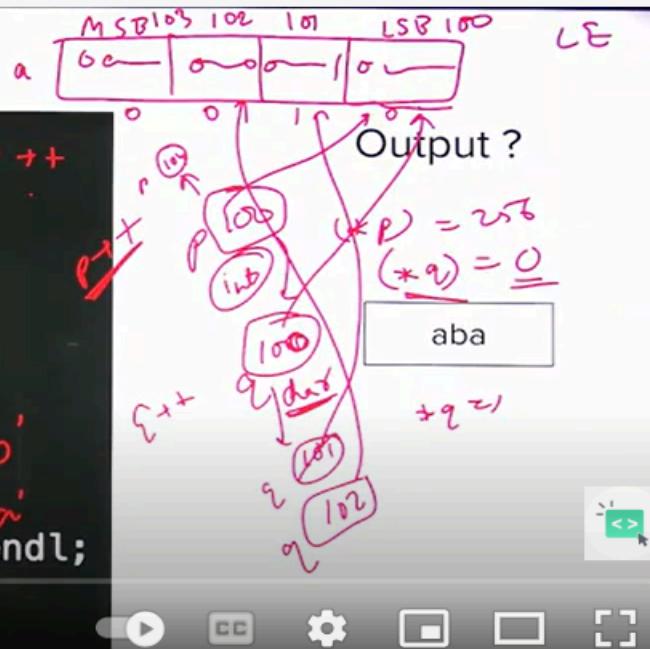
Example

```
#include<iostream>
using namespace std;
int main(){
    int a = 257;
    int *p = &a;
    cout<<*(char*)p + 'a'<<endl;
    cout<<p<<endl;
    return 0;
}
```

Output ?

Example

```
#include<iostream>
using namespace std; (*q)++
int main(){
    int a = 256;
    int *p = &a;
    char *q = (char*)p;
    cout<<char(*q+'a'); 'a'
    q++;
    cout<<char(*q+'a'); 'b'
    q++;
    cout<<char(*q+'a')<<endl;
    return 0;
}
```



Output depend on how the value are stored \Rightarrow little indian or big indian

Structures, Unions, and Enumerations | C++

Derived & User-Defined Data Types

Derived data types: The data-types that are derived from the primitive datatypes.

User-Defined data types: Data types that are defined by user itself.

- Arrays
- Pointers
- Structures
- Union
- Enum

Structures

A structure is a sequence of elements of arbitrary data type.

For example, if we want to store address,

```
struct Address {  
    const char* name;           // "Sameer"  
    int number;                // 45  
    const char* street;         // "Tilak Nagar"  
    const char* town;           // "Chennai"  
    char state[2];              // "Tamil Nadu"  
    const char* zip;            // "102039"  
};
```

Structures

- Let's see how to reference the elements:

- Using . (dot) operator:

```
Address add;  
add.name = "Sameer";  
add.number = 45;
```

- Through pointers using -> (struct pointer dereference) operator:

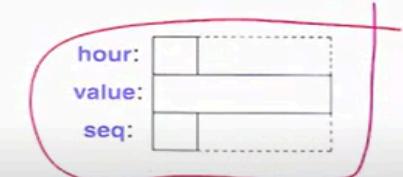
```
void prtAdd(Address* add){  
    cout << add->name;  
    cout << add->number; // and so on
```

Structures

- Total size of the structure is not always the sum of the sizes of all data members.

Let's understand this:

```
struct Readout{  
    char hour;  
    int value;  
    char seq;  
};
```



So the size of Readout on a 4 byte-int machine would be 12 bytes ~~and not 6 bytes~~, this is due to the padding added by the compiler to avoid alignment issues.

► alignment issues 31:21 / 1:09:44



Unions

- A **union** is a **struct** in which all members are allocated at the same address so that the union occupies only as much space as its largest member.
- It can only hold value for one member at a time.

For example, if we want to store address,

```
union Address {  
    const char* name;           // "Sameer"  
    int number;                 // 45  
    const char* street;         // "Tilak Nagar"  
};
```

union {
 int a; // 4B
 char b; // 1B
};

Unions

- Like structures, their members can also be accessed by:
 - Using . (dot) operator:
 - Through pointers using -> (union pointer dereference) operator:
- ```
Address add;
add.name = "Sameer";
add.number = 45; // and so on
```
- ```
void prtAdd(Address* add){  
    cout << add->name;  
    cout << add->number;    // and so on
```

Unions

- The size of a union is taken according to the size of largest member in it.

Let's understand this:

```
union Readout{  
    char hour;  
    int value;  
    char seq;  
};
```

So the size of Readout on a 4 byte-int machine would be 4 bytes and not 6 bytes, this is due to using same memory for all members.

Enumerations

- An enum is a type that can hold a set of integer values specified by the user.
- These values are defined at the time of declaring the enumerated type.

For example,

```
enum Color{  
    red,  
    green,  
    blue  
};
```

int c; →
color →
c = red
c = green
c = blue
c = 0 // 0 -> red

Here red, green and blue are called the enumerators with implicit values as 0, 1 and 2 respectively.

Enumerations

- We can also change the default value of an enum element at the time of declaration.

For example,

```
enum Color {  
    red = 10,  
    green = 20,  
    blue = 30  
};
```

Color c;
c = red ↔ c = 10

SL
(

Here red, green and blue have values 10, 20 and 30 respectively.

Enumerations

- Why enums are used ?
 - Because an enum variable takes only one value out of all possible values. This makes enums a good choice to work with flags.

For example,

```
enum light {
    red = 1,
    orange = 2,
    green = 3,
    yellow = 4
} signal;
```



Here the signal enum variable can take value out of 4 possible values

Examples

```
int main()
{
    struct var {
        int a, b;
    };
    struct var v;
    v.a=10;
    v.b=20;
    printf("%d\n", v.a);
    return 0;
}
```

- A. 10
B. 20
C. 30
D. 0

```
9 #include <iostream>
10
11 using namespace std;
12 int main()
13 {
14     union val
15     {
16         int a,b;
17     };
18     union val v;
19     v.a=10;
20     v.b=20;
21
22     cout<<v.a;
23     return 0;
24 }
```

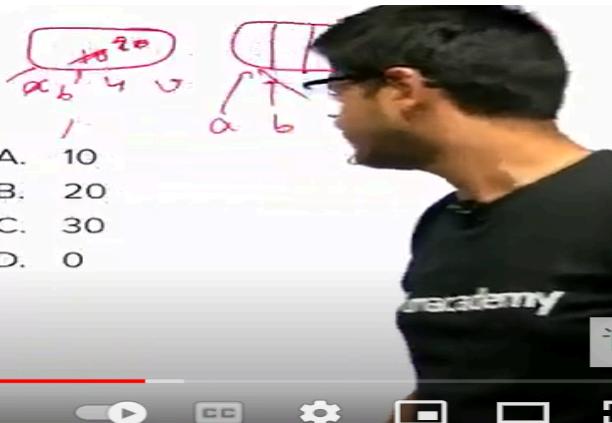
20

In union last variable will survive.

Examples

```
int main()
{
    union var {
        int a, b; char c;
    };
    union var v;
    v.a=10;
    v.b=20;
    printf("%d\n", v.a);
    return 0;
}
```

- A. 10
B. 20
C. 30
D. 0



Examples:

Q1:

Examples

```
int main() {
    enum days {MON=-1, TUE, WED=6, THU, FRI, SAT};
    printf("%d, %d, %d, %d, %d, %d\n", MON, TUE, WED, THU, FRI, SAT);
    return 0;
}
```

A. -1, 0, 1, 2, 3, 4
B. -1, 2, 6, 3, 4, 5
C. -1, 0, 6, 2, 3, 4
D. -1, 0, 0, 7, 8, 9

```
13 int main()
14 {
15     enum days{a,b,c=6,d,e,f};
16     cout<<a<<b<<c<<d<<e<<f<<endl;
17
18     return 0;
19 }
```

Next value will be prev_val+1;

By default → start from the 0

Q2:

Examples

The following C declarations

```
struct node
{
    int i;
    float j;
};

struct node *s[10];
define s to be?
```

A An array, each element of which is a pointer to a structure of type node

B A structure of 2 fields, each field being a pointer to an array of 10 elements

C A structure of 3 fields: an integer, a float, and an array of 10 elements

D An array, each element of which is a structure of type node

Q3:

The image shows three separate terminal windows side-by-side. Each window contains a C++ code snippet and its corresponding command-line output.

Left Window:

```
13 int main()
14 {
15     int a=1;
16     int b=1e4,c=1e4;
17     a=(b*c);
18     cout<<a<<endl;
19
```

Output: 1000000000

Middle Window:

```
13 int main()
14 {
15     int a=1;
16     int b=1e5,c=1e4;
17     a=(b*c);
18     cout<<a<<endl;
19
```

Output: 1000000000

Right Window:

```
13 int main()
14 {
15     int a=1;
16     int b=1e5,c=1e5;
17     a=(b*c);
18     cout<<a<<endl;
19
```

Output: 1410065408

Int can hold up to $1e9$;

$1e5 * 1e5 \Rightarrow$ overflow for the int

The image shows a screenshot of the CLion IDE interface. On the left is the project tree, and the main area displays the following C++ code:

```
2 using namespace std;
3 int main() {
4     int a;
5     int b = 1e5, c = 1e5;
6     a = b*c;
7     cout<<a<<endl;
8     return 0;
9 }
10 // int ~ 10^9
11 // binary of 10^10
12 // extract LS 4 bytes
```

At the bottom of the code editor, there is a status bar with the path: /Users/jaybansal/CLionProjects/CP/cmake-build-debug/CP and the output: 1410065408.

after $1e9 \rightarrow$ don't take int

long====> hold up to $1e18$

Q4:

The image shows three separate terminal windows side-by-side. Each window contains a C++ code snippet and its corresponding command-line output.

Left Window:

```
13 int main()
14 {
15     long a=1;
16     int b=1e5,c=1e5;
17     a=b*c;
18     cout<<a<<endl;
19
```

Output: 1410065408

Middle Window:

```
13 int main()
14 {
15     long a=1;
16     int b=1e5,c=1e5;
17     a=(long)b*(long)c;
18     cout<<a<<endl;
19
```

Output: 1410065408

Right Window:

```
13 int main()
14 {
15     long a=1;
16     int b=1e5,c=1e5;
17     a=(long)b*(long)c;
18     cout<<a<<endl;
19
20 }
```

Output: 1000000000000

Long = int*int \Rightarrow first do the int*int and then the result will be produced and then the result will be stored into the int(largest data type given for the calculation) and that result will be stored into the long.

