`_Alignas` is a type specifier that places storage boundaries for objects at multiples of the specified byte. For example, a `char` array of length 5 and `_Alignas` of 8 bytes will naturally occupy 5 bytes in memory but 8 bytes with alignment.

For convenience, the macro `alignas` can be replaced with `_Alignas`; the program then needs to include the `stdalign.h` header file as shown below:

```
#include <stdalign.h>
```

Syntax

`_Alignas` is written in the definition or declaration of an object or variable in any of the following ways:

- `_Alignas(expression)`
- `_Alignas(type)`

expression denotes a constant integral expression that evaluates to zero or a valid alignment.

type denotes a C type - e.g., int, float, double, etc.

Explanation

- `_Alignas` may only be specified on the declaration of a variable or the declaration/definition of a struct, union, and enumeration.
- The largest (and strictest) `_Alignas` value in the declaration specifies the alignment of an object.
- If the natural alignment of an object is larger than the specified alignment, natural alignment will apply.

The _Alignas specifier cannot be used in function parameters, typedef, exception parameters of a catch clause, or objects that are bit fields and have registered storage class.

Examples

The code below demonstrates the use of _Alignas.

- The program creates a struct with a float array of 4 elements.
- The natural size of this struct is 16 bytes, as a single float value occupies 4 bytes.
- The main program specifies an alignment of 32 bytes and creates two instances of the struct. The third instance is created without alignment.
- The print statements reveal addresses of all structs in memory. The byte difference between structs a and b is 32, which indicates that the alignment was successful. The byte difference between structs b and c is 16, which indicates natural alignment.

```
#include <stdalign.h>
#include <stdio.h>

// define a struct
struct align16
{
  float data[4];
};



int main(void)
{
  //align to 32 bytes
  alignas(32) struct align16 a,b;
  struct align16 c;
  //print the memory addresses.
  printf("%p\n", (void*)&a);
```

```c
  printf("%p\n", (void*)&b);
  printf("%p\n", (void*)&c);
}
```

RELATED TAGS

c

CONTRIBUTOR

Ayesha Naeem