C++ Linkage
External vs Internal

https://www.youtube.com/watch?v=TtWwtui5qqY

std : stone age
div : unknown

What is Linkage in C++?

- Describes the *accessibility of various objects* in C/C++ program,
  - from one file to another
  - or even within the same file

Programs are built up out of multiple *.CPP/C files and libraries (*.lib).

Thus, It is important to know, how objects in different files are referred from each other.

https://www.youtube.com/watch?v=TtWwtui5qqY

std : stone age
div : unknown

Understanding Linkage in C++

- For understanding Linkage, one need to have a good knowledge about what a C/C++ varaible is.

- In C/C++, a variable provides us with named storage, that our programs can manipulate(use & modify).

std : stone age
div : unknown

Understanding Linkage in C++

```
int nValue = 100;
static int nReferenceCount = 0;
extern int nTickCount;
```

what is the purpose of sepcifying 'static' ?

when exactly 'extern' is to be used ?

How varaibles are referenced from different files ?

std : stone age
div : unknown

Understanding Linkage in C++

- Linkage is a <u>property of</u> a <u>varaible name</u>

- The variable names used in different <u>translation unit</u> must follow certain rules.
  - a <u>translation unit</u> is the basic unit which are independently compilable by the compiler.
  - There can be several such translation units which are linked together to form a program.

std : stone age
div : unknown

Understanding Linkage in C++

During compilationm
Each Translation unit is complied to generate the corresponding object file(*.obj),

During link time,
All object files are processed by linker and generates machine code for application

https://www.youtube.com/watch?v=TtWwtui5qqY

Understanding Linkage in C++

std : stone age
div : unknown

- There are 3 types of linkage,
  1. No Linkage
  2. Internal linkage
  3. External linkage

std : stone age
div : unknown

Understanding Linkage in C++

- No Linkage
  - ➢ variables that are NOT explicitly declared extern or static;
  - ➢ local classes and their member functions;
  - ➢ other names declared at block scope such as typedefs, enumerations, and enumerators.

- Names not specified with external, or internal linkage also have no linkage, regardless of which scope they are declared in.

- Name of Variable with No-linkage is present only upto compilation
  I.e after compilation, there is no such name exists in program.

std : stone age
div : unknown

Understanding Linkage in C++

- No Linkage

Example

https://www.youtube.com/watch?v=TtWwtui5qqY

std : stone age
div : unknown

Understanding Linkage in C++

- Internal linkage
  - Here, the name can be referred to from all scopes in the current translation unit.
    - ➢ variables, functions, or function templates declared static
    - ➢ non-volatile, non-inline, const-qualified variables (including constexpr) that aren't declared extern and aren't previously declared to have external linkage;
    - ➢ data members of anonymous unions.
    - ➢ names declared in an unnamed namespace or in a namespace nested within an unnamed namespace

std : stone age
div : unknown

Understanding Linkage in C++

- Internal linkage

Example

https://www.youtube.com/watch?v=TtWwtui5qqY

Understanding Linkage in C++

- External linkage
  - Here, the name can be referred to from the scopes in the other translation units.
  - Any of the following names declared at namespace scope have external linkage
    - ➢ functions not declared static,
    - ➢ namespace-scope non-const variables not declared static,
    - ➢ and any variables declared extern

Note: if the name is declared in an unnamed namespace or in a namespace nested within an unnamed namespace, the name has internal linkage.

Understanding Linkage in C++

- External linkage

Example

std : stone age
div : unknown

Thank You

Subscribe to our YouTube Channel

LIKE & SHARE

https://www.youtube.com/watch?v=TtWwtui5qqY