

This lesson is on Functions

## OVERVIEW

The following are the coverage for Functions:

- Function Definition
- Function Prototypes
- Function Flow
- Parameter Passing: Call by Value
- Storage Scope of Variables
- Functional Decomposition

Content Copyright Nanyang Technological University

2

There are 6 main sections to cover for Functions. This video focused on the 5<sup>th</sup> topic: storage scope of variables

## LEARNING OBJECTIVES

Content Copyright Nanyang Technological University

3

## LEARNING OBJECTIVES

## LEARNING OBJECTIVES

At this lesson, you should be able to:

At this lesson, you should be able to

## LEARNING OBJECTIVES

At this lesson, you should be able to:

- Discuss the storage scope of variables

Discuss the storage scope of variables

## LEARNING OBJECTIVES

At this lesson, you should be able to:

- Discuss the storage scope of variables
- Identify local and global variables and use them appropriately in a program

Identify local and global variables and use appropriately in a program

**STORAGE CLASS**

- The storage class of a variable is a set of properties about the variable.

Content Copyright Nanyang Technological University 7

The storage class of a variable is a set of properties about the variable

## STORAGE CLASS

- The storage class of a variable is a set of properties about the variable.
- It is determined by where it is defined and which keyword (i.e. **auto**, **extern**, **static** or **register**) it is used with.

Content Copyright Nanyang Technological University 8

It is determined by where it is defined and which keyword (i.e. **auto**, **extern**, **static** or **register**) it is used with.

## STORAGE CLASS

- The storage class of a variable is a set of properties about the variable.
- It is determined by where it is defined and which keyword (i.e. **auto**, **extern**, **static** or **register**) it is used with.
- The storage class of a variable determines the scope, linkage and storage duration of the variable.

Content Copyright Nanyang Technological University 9

The storage class of a variable determines the scope, linkage and storage duration of the variable.

## STORAGE CLASS

- The *scope* determines the region of the code that can use the variable. In other words, the variable is visible in that section of code.

Content Copyright Nanyang Technological University

10

The scope determines the region of the code that can use the variable. In other words, the variable is visible in that section of code.

## STORAGE CLASS

- The *scope* determines the region of the code that can use the variable. In other words, the variable is visible in that section of code.
- The *linkage* determines how a variable can be used in a multiple-source file program. It identifies whether the variable can only be used in the current source file or it can be used in other source files with proper declarations.

Content Copyright Nanyang Technological University

11

The *linkage* determines how a variable can be used in a multiple-source file program. It identifies whether the variable can only be used in the current source file or it can be used in other source files with proper declarations.

## STORAGE CLASS

- The *scope* determines the region of the code that can use the variable. In other words, the variable is visible in that section of code.
- The *linkage* determines how a variable can be used in a multiple-source file program. It identifies whether the variable can only be used in the current source file or it can be used in other source files with proper declarations.
- The *storage duration* determines how long a variable can exist in memory.

Content Copyright Nanyang Technological University 12

The storage duration of a C variable can be static or automatic. The *storage duration* determines how long a variable can exist in memory.

## SCOPE OF VARIABLES IN A FUNCTION

```
float areaOfCircle(float radius) { // parameter
    const float pi = 3.14;           // const variable
    float area;                    // local variable
    area = pi*radius*radius;
    return area;
}
```

Example above: Variables radius, pi and area are NOT visible outside this function.

Content Copyright Nanyang Technological University 13

The variables radius, pi and area in the program shown are NOT visible outside this function.

## SCOPE OF VARIABLES IN A FUNCTION

```
float areaOfCircle(float radius) { // parameter
    const float pi = 3.14;           // const variable
    float area;                    // local variable
    area = pi*radius*radius;
    return area;
}
```

Example above: Variables radius, pi and area are NOT visible outside this function.

Variables declared in a function are ONLY visible within that function. We call it block scope.

Content Copyright Nanyang Technological University 14

Variables declared in a function is ONLY visible within that function. We call it block scope.

## STATIC VARIABLES

- A *static* variable may be defined inside or outside a function's body.

Content Copyright Nanyang Technological University 15

### Static Variables

A *static* variable may be defined inside or outside a function's body.

## STATIC VARIABLES

- A *static* variable may be defined inside or outside a function's body.
- The duration of a static variable is fixed. Static variables are created at the start of the program and are destroyed only at the end of the program execution.

Content Copyright Nanyang Technological University 16

The duration of a static variable is fixed. Static variables are created at the start of the program and are destroyed only at the end of the program execution.

## STATIC VARIABLES

- A *static* variable may be defined inside or outside a function's body.
- The duration of a static variable is fixed. Static variables are created at the start of the program and are destroyed only at the end of the program execution.
- We can define static variables *inside* a function's body by changing an automatic variable using the keyword **static**.

Content Copyright Nanyang Technological University 17

We can define static variables *inside* a function's body by changing an automatic variable using the keyword **static**.

## STATIC VARIABLES

- If a static variable is defined and initialised, it is then initialised once when the storage is allocated.

Content Copyright Nanyang Technological University

18

If a static variable is defined and initialised, it is then initialised once when the storage is allocated.

## STATIC VARIABLES

- If a static variable is defined and initialised, it is then initialised once when the storage is allocated.
- If a static variable is defined, but not initialised, it will be initialised to zero by the compiler. The initialisation is done when the storage is allocated.

Content Copyright Nanyang Technological University

19

If a static variable is defined, but not initialised, it will be initialised to zero by the compiler. The initialisation is done when the storage is allocated.

## STATIC VARIABLES

- If a static variable is defined and initialised, it is then initialised once when the storage is allocated.
- If a static variable is defined, but not initialised, it will be initialised to zero by the compiler. The initialisation is done when the storage is allocated.
- If the static variable is defined inside a function's body, then the variable is only visible by the block containing the variable.

Content Copyright Nanyang Technological University

20

If the static variable is defined inside a function's body, then the variable is only visible by the block containing the variable.

## STATIC VARIABLES

- Static variables are very useful when we need to write functions that retain values between functions.

Content Copyright Nanyang Technological University

21

Static variables are very useful when we need to write functions that retain values between functions.

## STATIC VARIABLES

- Static variables are very useful when we need to write functions that retain values between functions.
- We may use global variables to achieve the same purpose.

Content Copyright Nanyang Technological University

22

We may use global variables to achieve the same purpose

## STATIC VARIABLES

- Static variables are very useful when we need to write functions that retain values between functions.
- We may use global variables to achieve the same purpose.
- However, static variables are preferable as they are local variables to the functions, and the shortcomings of global variables can be avoided.

Content Copyright Nanyang Technological University

23

However, static variables are preferable as they are local variables to the functions, and the shortcomings of global variables can be avoided.

## AUTOMATIC VARIABLES

- Automatic (or **auto**) variables are declared with the storage class keyword **auto** inside the body of a function or block.

Content Copyright Nanyang Technological University 24

Automatic (or **auto**) variables are declared with the storage class keyword **auto** inside the body of a function or block.

## AUTOMATIC VARIABLES

- Automatic (or **auto**) variables are declared with the storage class keyword **auto** inside the body of a function or block.
- A block contains a complex statement that is enclosed by braces **{}**.

Content Copyright Nanyang Technological University 25

A block contains a complex statement that is enclosed by braces

## AUTOMATIC VARIABLES

- Automatic (or **auto**) variables are declared with the storage class keyword **auto** inside the body of a function or block.
- A block contains a complex statement that is enclosed by braces {}.
- We can define an automatic variable by using the storage class keyword **auto** as:  
**auto int i, j, k;**

Content Copyright Nanyang Technological University 26

We can define an automatic variable by using the storage class keyword **auto** as **auto int i, j, k**

## AUTOMATIC VARIABLES

- Automatic (or **auto**) variables are declared with the storage class keyword **auto** inside the body of a function or block.
- A block contains a complex statement that is enclosed by braces {}.
- We can define an automatic variable by using the storage class keyword **auto** as:  
**auto int i, j, k;** or  
we can omit the keyword as: **int i, j, k;**

Content Copyright Nanyang Technological University 27

or we can omit the keyword auto.

## AUTOMATIC VARIABLES

- An automatic variable has automatic storage. The lifetime of an automatic variable is only within the function or block where it is defined.

Content Copyright Nanyang Technological University 28

An automatic variable has automatic storage. The lifetime of an automatic variable is only within the function or block where it is defined.

## AUTOMATIC VARIABLES

- An automatic variable has automatic storage. The lifetime of an automatic variable is only within the function or block where it is defined.
- It has no meaning outside the function or block. It is also called the *local variable* of the corresponding function or block.

Content Copyright Nanyang Technological University 29

It has no meaning outside the function or block. It is also called the *local variable* of the corresponding function or block.

## AUTOMATIC VARIABLES

- An automatic variable has automatic storage. The lifetime of an automatic variable is only within the function or block where it is defined.
- It has no meaning outside the function or block. It is also called the *local variable* of the corresponding function or block.
- Automatic variables are destroyed after the execution of the corresponding function or block where they are defined.

Content Copyright Nanyang Technological University 30

Automatic variables are destroyed after the execution of the corresponding function or block where they are defined. It

## AUTOMATIC VARIABLES

- It is because they are no longer needed and the memory occupied by the variables can be reused by other functions.

Content Copyright Nanyang Technological University 31

It is because they are no longer needed and the memory occupied by the variables can be reused by other functions.

## AUTOMATIC VARIABLES

- It is because they are no longer needed and the memory occupied by the variables can be reused by other functions.
- Therefore, the value stored in an automatic variable is lost after exiting from the function or block. An automatic variable has the block scope.

Content Copyright Nanyang Technological University 32

Therefore, the value stored in an automatic variable is lost after exiting from the function or block. An automatic variable has the block scope.

## AUTOMATIC VARIABLES

- It is because they are no longer needed and the memory occupied by the variables can be reused by other functions.
- Therefore, the value stored in an automatic variable is lost after exiting from the function or block. An automatic variable has the block scope.
- Only the function in which the variable is defined can access that variable by name. Further, an automatic variable has no linkage.

Content Copyright Nanyang Technological University

33

Only the function in which the variable is defined can access that variable by name. Further, an automatic variable has no linkage.

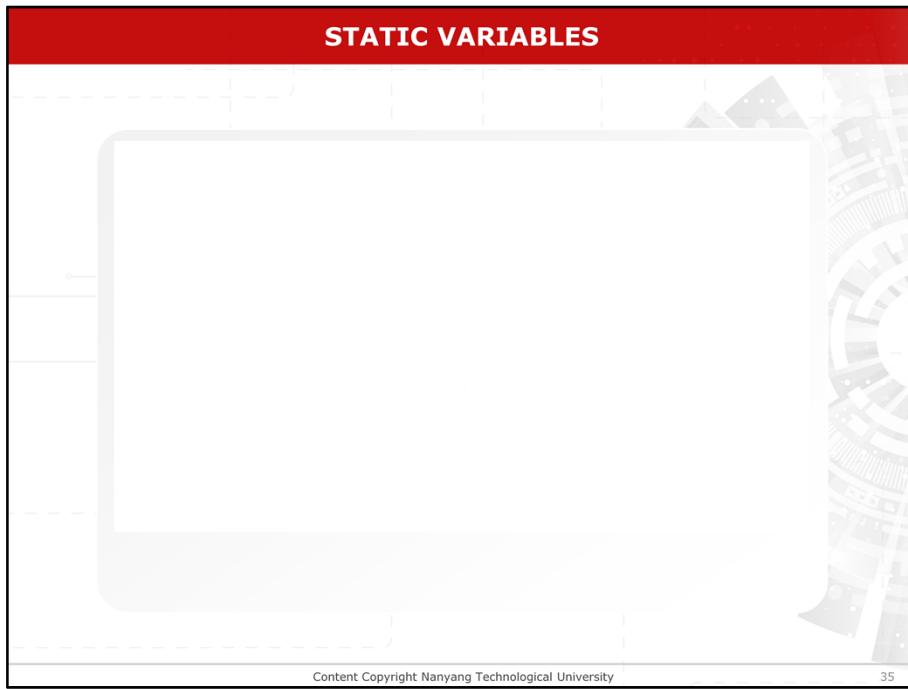
## AUTOMATIC VARIABLES

- It is because they are no longer needed and the memory occupied by the variables can be reused by other functions.
- Therefore, the value stored in an automatic variable is lost after exiting from the function or block. An automatic variable has the block scope.
- Only the function in which the variable is defined can access that variable by name. Further, an automatic variable has no linkage.
- The variable cannot be declared twice in the same block.

Content Copyright Nanyang Technological University

34

The variable cannot be declared twice in the same block.



The number of static and automatic variables are computed as shown. Automatic variable or local variables disappear after each function execution. Static variable like global variables stay until end of program execution.

## EXTERNAL VARIABLES

- An external (or **extern**) variable is defined outside a function's body.

Content Copyright Nanyang Technological University 36

### External Variables

An external (or **extern**) variable is defined outside a function's body.

## EXTERNAL VARIABLES

- An external (or **extern**) variable is defined outside a function's body.
- However, it can also be declared inside a function that uses it by using the **extern** keyword.

Content Copyright Nanyang Technological University 37

However, it can also be declared inside a function that uses it by using the **extern** keyword.

## EXTERNAL VARIABLES

- An external (or **extern**) variable is defined outside a function's body.
- However, it can also be declared inside a function that uses it by using the **extern** keyword.
- External variables have static storage duration and external linkage.

Content Copyright Nanyang Technological University 38

External variables have static storage duration and external linkage.

## EXTERNAL VARIABLES

- An external (or **extern**) variable is defined outside a function's body.
- However, it can also be declared inside a function that uses it by using the **extern** keyword.
- External variables have static storage duration and external linkage.
- The storage is allocated to the variable when it is declared and it lasts until the end of program execution.

Content Copyright Nanyang Technological University 39

The storage is allocated to the variable when it is declared and it lasts until the end of program execution.

## EXTERNAL VARIABLES

- An external (or **extern**) variable is defined outside a function's body.
- However, it can also be declared inside a function that uses it by using the **extern** keyword.
- External variables have static storage duration and external linkage.
- The storage is allocated to the variable when it is declared and it lasts until the end of program execution.
- If the variable is defined in another file, declaring the variable with the keyword **extern** is mandatory.

Content Copyright Nanyang Technological University 40

If the variable is defined in another file, declaring the variable with the keyword **extern** is mandatory.

## EXTERNAL VARIABLES

- External variables are initialised once when the storage is allocated.

Content Copyright Nanyang Technological University 41

External variables are initialised once when the storage is allocated.

## EXTERNAL VARIABLES

- External variables are initialised once when the storage is allocated.
- If the external variable is not explicitly initialised in the program, it will be automatically initialised to zero by the compiler.

Content Copyright Nanyang Technological University 42

If the external variable is not explicitly initialised in the program, it will be automatically initialised to zero by the compiler.

## EXTERNAL VARIABLES

- External variables are initialised once when the storage is allocated.
- If the external variable is not explicitly initialised in the program, it will be automatically initialised to zero by the compiler.
- External variables allow us to break down a large program into smaller files and compile each file separately.

Content Copyright Nanyang Technological University

43

External variables allow us to break down a large program into smaller files and compile each file separately.

## EXTERNAL VARIABLES

- External variables are initialised once when the storage is allocated.
- If the external variable is not explicitly initialised in the program, it will be automatically initialised to zero by the compiler.
- External variables allow us to break down a large program into smaller files and compile each file separately.
- This helps to reduce program development time, as only those files that have been changed are required to be compiled again.

Content Copyright Nanyang Technological University

44

This helps to reduce program development time, as only those files that have been changed are required to be compiled again.

**REGISTER VARIABLES**

- An automatic variable can be defined using the keyword **register**.

Content Copyright Nanyang Technological University 45

An automatic variable can be defined using the keyword **register**.

## REGISTER VARIABLES

- An automatic variable can be defined using the keyword **register**.
- It informs the compiler that the variables will be made reference to on numerous occasions and, where possible, to use the CPU registers.

Content Copyright Nanyang Technological University 46

It informs the compiler that the variables will be made reference to on numerous occasions and, where possible, to use the CPU registers.

## REGISTER VARIABLES

- An automatic variable can be defined using the keyword **register**.
- It informs the compiler that the variables will be made reference to on numerous occasions and, where possible, to use the CPU registers.
- Instructions using register variables execute faster than instructions that use no register variables.

Content Copyright Nanyang Technological University 47

Instructions using register variables execute faster than instructions that use no register variables.

## REGISTER VARIABLES

- An automatic variable can be defined using the keyword **register**.
- It informs the compiler that the variables will be made reference to on numerous occasions and, where possible, to use the CPU registers.
- Instructions using register variables execute faster than instructions that use no register variables.
- However, only a limited number of registers are available.

Content Copyright Nanyang Technological University 48

However, only a limited number of registers are available.

## REGISTER VARIABLES

- The use of register variables is applicable to automatic variables and function arguments only and is restricted to certain data types (which is machine dependent) such as **int** and **char**.

Content Copyright Nanyang Technological University 49

The use of register variables is applicable to automatic variables and function arguments only and is restricted to certain data types (which is machine dependent) such as **int** and **char**.

## REGISTER VARIABLES

- The use of register variables is applicable to automatic variables and function arguments only and is restricted to certain data types (which is machine dependent) such as **int** and **char**.
- Pointers are not allowed to take the address of register variables, i.e. **&** operator will not work with register variables.

Content Copyright Nanyang Technological University 50

Pointers are not allowed to take the address of register variables, i.e. **&** operator will not work with register variables.

## STATIC VARIABLES

- It is defined using the static keyword.

Content Copyright Nanyang Technological University

51

## STATIC VARIABLES

- It is defined using the **static** keyword.
- The duration of a static variable is fixed.

Content Copyright Nanyang Technological University

52

## STATIC VARIABLES

- It is defined using the **static** keyword.
- The duration of a static variable is fixed.
- Static variables are created at the **start** of the program and are destroyed only at the **end** of program execution.

Remove the

Content Copyright Nanyang Technological University

53

## STATIC VARIABLES

- It is defined using the **static** keyword.
- The duration of a static variable is fixed.
- Static variables are created at the **start** of the program and are destroyed only at the **end** of program execution.
- That is, they exist throughout the program execution once they are created.

Remove the

**LOCAL VARIABLES**

- Local variables are variables defined inside a function.

Content Copyright Nanyang Technological University 55

### Local and Global Variables

Local variables are variables defined inside a function. They have function or block scope. They can be accessed only within the function. They cannot be accessed by other functions. Local variables are created when the function is invoked, and destroyed after the complete execution of the function.

## LOCAL VARIABLES

- Local variables are variables defined inside a function.
- They have function or block scope. They can be accessed only within the function. They cannot be accessed by other functions.

Content Copyright Nanyang Technological University 56

### Local and Global Variables

Local variables are variables defined inside a function. They have function or block scope. They can be accessed only within the function. They cannot be accessed by other functions. Local variables are created when the function is invoked, and destroyed after the complete execution of the function.

## LOCAL VARIABLES

- Local variables are variables defined inside a function.
- They have function or block scope. They can be accessed only within the function. They cannot be accessed by other functions.
- Local variables are created when the function is invoked, and destroyed after the complete execution of the function.

Content Copyright Nanyang Technological University 57

### Local and Global Variables

Local variables are variables defined inside a function. They have function or block scope. They can be accessed only within the function. They cannot be accessed by other functions. Local variables are created when the function is invoked, and destroyed after the complete execution of the function.

## GLOBAL VARIABLES

- Global variables are variables defined outside the functions.

Content Copyright Nanyang Technological University

58

Global variables are variables defined outside the functions

## GLOBAL VARIABLES

- Global variables are variables defined outside the functions.
- They have file (or program) scope.

Content Copyright Nanyang Technological University 59

They have file or program scope.

## GLOBAL VARIABLES

- Global variables are variables defined outside the functions.
- They have file (or program) scope.
- Thus, global variables are visible to all the functions that are defined following its declaration.

Content Copyright Nanyang Technological University 60

Thus, global variables are visible to all the functions that are defined following its declaration.

## ADVANTAGES OF GLOBAL VARIABLES

The advantages of global variables in programs are that global variables are the simplest way of communication between functions and they are efficient.

Content Copyright Nanyang Technological University

61

The advantages of global variables in programs are that global variables are the simplest way of communication between functions and they are efficient.

## DISADVANTAGES OF GLOBAL VARIABLES

- They are less readable and more difficult to debug and modify as any functions in the program can change the value of the global variable.

Content Copyright Nanyang Technological University

62

The disadvantages of programs using global variables are that they are less readable and more difficult to debug and modify as any functions in the program can change the value of the global variable. Therefore, it is a good programming practice to use local variables, and use parameter passing between functions for communication between functions. In this way, the value of each variable in the function is protected.

## DISADVANTAGES OF GLOBAL VARIABLES

- They are less readable and more difficult to debug and modify as any functions in the program can change the value of the global variable.
- Therefore, it is a good programming practice to use local variables, and use parameter passing between functions for communication between functions.

Content Copyright Nanyang Technological University

63

The disadvantages of programs using global variables are that they are less readable and more difficult to debug and modify as any functions in the program can change the value of the global variable. Therefore, it is a good programming practice to use local variables, and use parameter passing between functions for communication between functions. In this way, the value of each variable in the function is protected.

## DISADVANTAGES OF GLOBAL VARIABLES

- They are less readable and more difficult to debug and modify as any functions in the program can change the value of the global variable.
- Therefore, it is a good programming practice to use local variables, and use parameter passing between functions for communication between functions.
- In this way, the value of each variable in the function is protected.

Content Copyright Nanyang Technological University

64

The disadvantages of programs using global variables are that they are less readable and more difficult to debug and modify as any functions in the program can change the value of the global variable. Therefore, it is a good programming practice to use local variables, and use parameter passing between functions for communication between functions. In this way, the value of each variable in the function is protected.

## LOCAL AND GLOBAL VARIABLES: EXAMPLE

```
#include <stdio.h>
int g_var = 5;
int fn1(int, int);
float expn(float);
int main( ) {
    char reply;
    int num;
    ...
}
int fn1(int x, int y) {

    float fnum;
    int temp;
    g_var += 10;
    ...
}
float expn(float n) {
    float temp;
    ...
}
```

Content Copyright Nanyang Technological University

65

This program illustrates the use of local and global variables

## LOCAL AND GLOBAL VARIABLES: EXAMPLE

```
#include <stdio.h>
int g_var = 5;           // global variable - has file scope
int fn1(int, int);
float expn(float);
int main( ) {
    char reply;
    int num;
    ...
}
int fn1(int x, int y) {
    float fnum;
    int temp;
    g_var += 10
    ...
}
float expn(float n) {
    float temp;
    ...
}
```

Content Copyright Nanyang Technological University 66

The global variable **g\_var** is declared outside the `main()` function. Global variables will have the file scope.

## LOCAL AND GLOBAL VARIABLES: EXAMPLE

```
#include <stdio.h>
int g_var = 5;
int fn1(int, int);
float expn(float);
int main() {
    char reply;           // local - These two variables are only
    int num;             // known inside main() function - block scope
    ...
}
int fn1(int x, int y) {

    float fnum;
    int temp;
    g_var += 10;
    ...
}
float expn(float n) {
    float temp;
    ...
}
```

Content Copyright Nanyang Technological University

67

- these two variables are only known inside main() function, they are said to be block scope

## LOCAL AND GLOBAL VARIABLES: EXAMPLE

```
#include <stdio.h>
int g_var = 5;
int fn1(int, int);
float expn(float);
int main() {
    char reply;
    int num;
    ...
}
int fn1(int x, int y) {      // local x,y - formal parameters are only
    float fnum;              // known inside this function – Block scope
    int temp;
    g_var += 10;
    ...
}
float expn(float n) {
    float temp;
    ...
}
```

Content Copyright Nanyang Technological University

68

- these two variables are only known inside this function, hence they are said to be block scope

## LOCAL AND GLOBAL VARIABLES: EXAMPLE

```
#include <stdio.h>
int g_var = 5;
int fn1(int, int);
float expn(float);
int main() {
    char reply;
    int num;
    ...
}
int fn1(int x, int y) {
    float fnum; // local - These two variables are known
    int temp;   // in this function only. - Block scope
    g_var += 10;
    ...
}
float expn(float n) {
    float temp // local - this variable is known in expn()
    ...
}
```

Content Copyright Nanyang Technological University

69

The variables **fnum** and **temp** are local variables which will have block scope.

**SUMMARY**

After this lesson, you should be able to:

- Discuss the storage scope of variables
- Identify local and global variables and use them appropriately in a program

Content Copyright Nanyang Technological University 70

In summary, after viewing this video lesson, you should be able to do the listed