Programming Language-II

Lecture # 7

Lecture Content

- Storage classes
  - Automatic(local)
  - External(global)
  - Static
  - register

Storage class

- A storage class defines the scope (visibility) and life-time of variables and/or functions within a C++ Program. These specifies precede the type that they modify.

auto Storage Class

- The **auto** storage class is the default storage class for all local variables.

```
int main()
{
int sum, num;
}
```

- The example above defines two variables with the same storage class, auto can only be used within functions, i.e., local variables.

Example 1

```
void func1() {
int x = 4;
cout << x << endl;
}

void func2() {
int x = 5;
cout << x << endl;
}

int main() {
func1();
func2();
return 0;
}
```

Example 2

```
void test();
int main(){
// local variable to main()
int var = 5;
test();
// illegal: var1 not declared inside main()
var1 = 9;
}
void test(){
// local variable to test()
int var1;
var1 = 6;
// illegal: var not declared inside test()
cout << var;
}
```

Global variables

- Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the life-time of your program.
- A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables

Global variable example 1

```
#include <iostream>
using namespace std;

int g;

int main() {
int a = 4;
g = a * 2;
cout << g << endl;
return 0;
}
```

Global variable example 2

A program can have same name for local and global variables but value of local variable inside a function will take preference. For example,

```
#include <iostream>
using namespace std;
int g = 10;
void func1() {
g = 20;
cout << g << endl;
}
int main() {
func1();
g = 30;
cout << g << endl;
return 0;
}
```

static Storage Class

- The **static** storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

static Storage Class(Cont…)

- The static modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared.

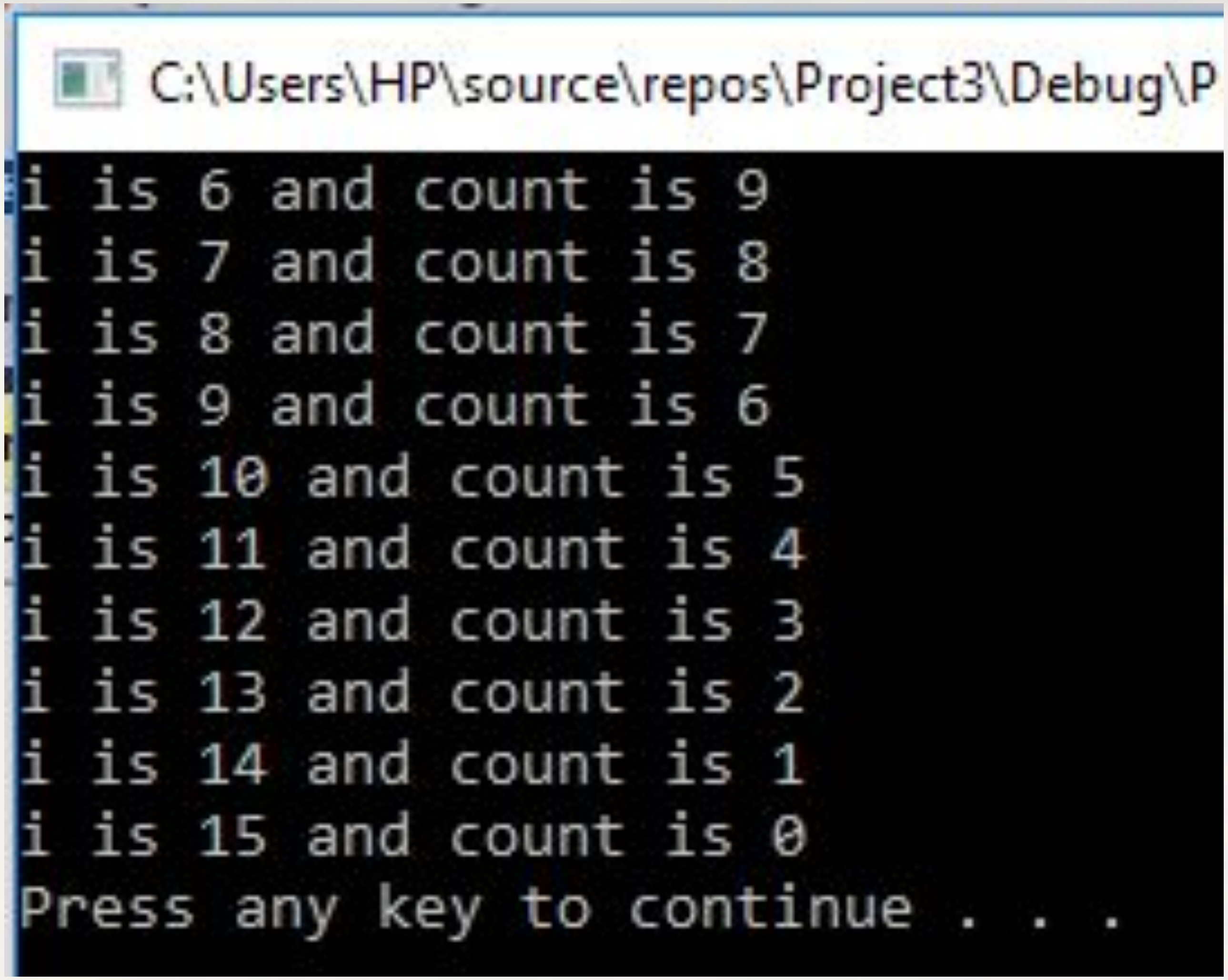
```
int fun(){
static int count = 0;
count++;
return count;
}

int main(){
cout<< fun();
cout<< fun();
return 0;
}
```

Example

```
#include <iostream>
using namespace std;
int count = 10;
void func();
int main() {
while (::count--) {
func();}
system("pause");
return 0;
}
void func() {
static int i = 5; // local static variable
i++;
cout << "i is " << i;
cout << " and count is " << ::count << endl;
}
```

Output



C:\Users\HP\source\repos\Project3\Debug\P

```
i is 6 and count is 9
i is 7 and count is 8
i is 8 and count is 7
i is 9 and count is 6
i is 10 and count is 5
i is 11 and count is 4
i is 12 and count is 3
i is 13 and count is 2
i is 14 and count is 1
i is 15 and count is 0
Press any key to continue . . .
```

register Storage Class

1. register keyword is used to define local variable.
2. Local variable are stored in register instead of **RAM**.
3. As variable is stored in register, the **Maximum size of variable = Maximum Size of Register**
4. unary operator [&] is not associated with it because Value is not stored in RAM instead it is stored in Register.
5. This is generally used for **faster access**.
6. Common use is “**Counter**“

```
int main(){
int num1, num2;
register int sum;

cout<<"\nEnter the Number 1 : ";
cin>>num1;
cout<<"\nEnter the Number 2 : ";
cin>>num2;
sum = num1 + num2;
cout<<sum;
return(0);
}
```

register Storage Class(Cont…)

- The register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions.

Initializing Local and Global Variables

- When a local variable is defined, it is not initialized by the system, you must initialize it yourself. Global variables are initialized automatically by the system when you define them as follows,

Data Type	Initializer
int	0
char	'\0'
float	0
double	0
pointer	NULL