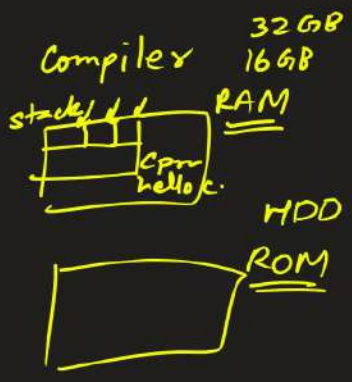
C Programming Lecture 3

Tuesday, 11 June 2024 8:16 PM

Types of Variables in C

Stacks
heaps



Local Variable ✓

A variable that is declared inside the function or block is called a local variable.

```
void function1() {
    int x=10; //local variable
}
```

Storage: Stack, Default Initial: Garbage, Scope: Within block, Life: End of the block

Global Variable

A variable that is declared outside the function or block is called a global variable. Any function can change the value of the global variable. It is available to all the functions.

```
int value=20; //global variable ✓
void function1() {
    int x=10; //local variable ✓
}
```

```
int a;
void main() {
    printf("%d", a);
}
```

0

Storage: Data segment, Default Initial: 0, Scope: Within the program, Life: End of the program

3

Automatic Variable

All variables in C that are declared inside the block, are automatic variables by default. We can explicitly declare an automatic variable using **auto keyword**.

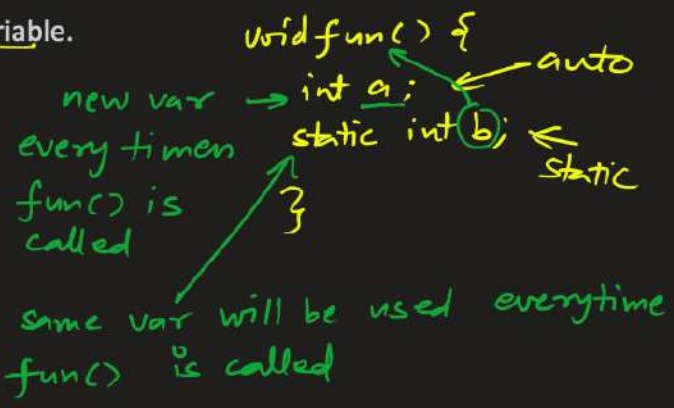
```
void main(){
    int x=10;//local variable (also automatic)
    auto int y=20;//automatic variable
}
```

Storage: Stack, Default Initial: Garbage, Scope: Within block, Life: End of the block

Static Variable

A variable that is declared with the static keyword is called static variable.
It retains its value between multiple function calls.

```
void function1(){
    int x=10;//local variable
    static int y=10;//static variable
    x=x+1;
    y=y+1;
    printf("%d,%d",x,y);
}
```



If you call this function many times, the **local variable will print the same value** for each function call, e.g, 11,11,11 and so on. But the **static variable will print the incremented value** in each function call, e.g. 11, 12, 13 and so on.
Storage: Data segment, Default Initial: 0, Scope: Within block, Life: End of the program

External Variable

We can share a variable in multiple C source files by using an external variable. To declare an external variable, you need to use **extern keyword**.

myfile.h

```
int x=10; //external variable (also global)
```

program1.c

```
#include "myfile.h"
#include <stdio.h>
extern int x;
void printValue(){
    printf("External variable: %d", x);
}
```

Storage: Data segment, Default Initial: 0, Scope: Entire Program, Life: End of the program

Data Types in C

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.

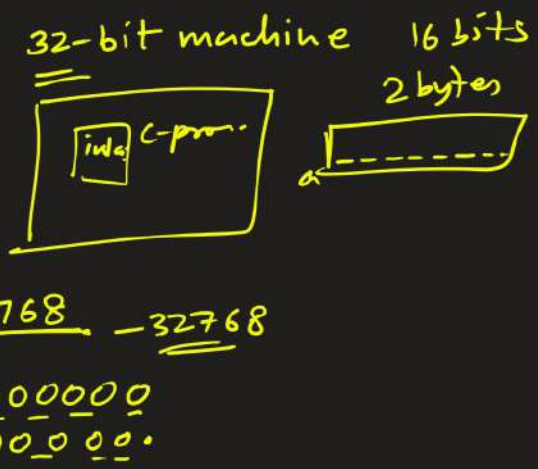
There are the following data types in C language.

Types	Data Types
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union
Enumeration Data Type	enum
Void Data Type	void

Basic Data Types

- The basic data types are integer-based and floating-point based. C language supports both signed and unsigned literals.
- The memory size of the basic data types may change according to 32 or 64-bit operating system.

Data Types	Memory Size	Range
char	1 byte	–128 to 127
signed char	1 byte	–128 to 127
unsigned char	1 byte	0 to 255
short	2 byte	–32,768 to 32,767
signed short	2 byte	–32,768 to 32,767
unsigned short	2 byte	0 to 65,535
int	2 byte	–32,768 to 32,767
signed int	2 byte	–32,768 to 32,767
unsigned int	2 byte	0 to 65,535
short int	2 byte	–32,768 to 32,767
signed short int	2 byte	–32,768 to 32,767
unsigned short int	2 byte	0 to 65,535
long int	4 byte	-2,147,483,648 to 2,147,483,647
signed long int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned long int	4 byte	0 to 4,294,967,295
float	4 byte ✓	
double	8 byte ✓	
long double	10 byte ✓	



32768 -32768

1000000
000000

Derived Data Types

Beyond the fundamental data types, C also supports ***derived data types,*** including ***arrays, pointers, structures,*** and ***unions***. These data types give programmers the ability to handle heterogeneous data, directly modify memory, and build complicated data structures.

Array

An ***array, a derived data type***, lets you store a sequence of ***fixed-size elements*** of the same type. It provides a mechanism for joining multiple targets of the same data under the same name.
The index is used to access the elements of the array, with a ***0 index*** for the first entry. The size of the array is fixed at declaration time and cannot be changed during program execution. The array components are placed in adjacent memory regions.

```
int numbers[5]; // Declares an integer array with a size of 5 elements
```

Pointer

A ***pointer*** is a derived data type that keeps track of another data type's memory address. When a ***pointer*** is declared, the ***data type*** it refers to is ***stated first***, and then the ***variable name*** is preceded by ***an asterisk (*).***
You can have incorrect access and change the value of variable using pointers by specifying the memory address of the variable. ***Pointers*** are commonly used in ***tasks*** such as ***function pointers, data structures***, and ***dynamic memory allocation***.

```
int *ptr;      // Declares a pointer to an integer
```

Structure

A structure is a derived data type that enables the creation of composite data types by allowing the grouping of many data types under a single name. It gives you the ability to create your own unique data structures by fusing together variables of various sorts.

```
// Define a structure representing a person
struct Person {
    char name[50];
and int age;
and float height;
};
```

character array
integer
floating point

50×1 + 1×2 + 1×4 = 56 Bytes

int a;

Person p;

Union

A derived data type called a ***union*** enables you to store various data types in the same memory address. In contrast to structures, where each member has a separate memory space, members of a union all share a single memory space. A value can only be held by one member of a union at any given moment.

```
// Define a union representing a numeric value
union NumericValue {
    int intValue;
or  float floatValue;
or  char stringValue[20];
};
```

20B
8B
20B

only one of them active

one of

Enumeration Data Type

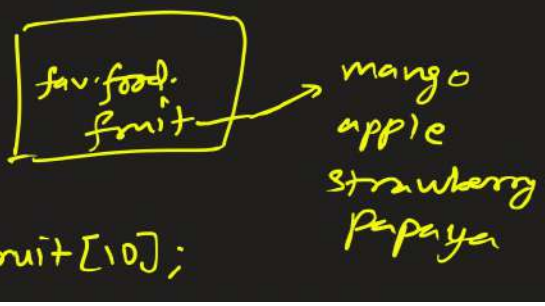
A set of named constants or ***enumerators*** that represent a collection of connected values can be defined in C using the ***enumeration data type (enum)***. ***Enumerations*** give you the means to give names that make sense to a group of integral values, which makes your code easier to read and maintain.

```
enum fruits{mango, apple, strawberry, papaya};
```

0 1 2 3

The default value of mango is 0, apple is 1, strawberry is 2, and papaya is 3. If we want to change these default values, then we can do as given below:

```
enum fruits{
mango=2,
apple=1,
strawberry=5,
papaya=7,
};
```



```c
// Define an enumeration for days of the week
enum DaysOfWeek {
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday
};

int main()  {
    enum DaysOfWeek w; // variable declaration of DaysOfWeek type
    w = Monday; // assigning value of Monday to w.
    printf("The value of w is %d",w);
    return 0;
}
```

int day; → -32k to 32k

DaysOfWeek

- The enum names available in an enum type can have the same value.
- If we do not provide any value to the enum names, then the compiler will automatically assign the default values to the enum names starting from 0.
- We can also provide the values to the enum name in any order, and the unassigned names will get the default value as the previous one plus one.
- The values assigned to the enum names must be integral constant, i.e., it should not be of other types such string, float, etc.
- All the enum names must be unique in their scope, i.e., if we define two enum having same scope, then these two enums should have different enum names otherwise compiler will throw an error.
- In enumeration, we can define an enumerated data type without the name also.

```c
int main(void) {
    enum fruits{mango = 1, strawberry=0, apple=1};
    printf("The value of mango is %d", mango);
    printf("\nThe value of apple is %d", apple);
    return 0;
}
```

```c
enum status{success, fail};
enum boolen{fail,pass};
int main(void) {
    printf("The value of success is %d", success);
    return 0;
}
```

```c
enum {success, fail} status;
int main() {
    status=success;
    printf("The value of status is %d", status);
    return 0;
}
```

Void Data Type

The ***void data type*** in the C language is used to denote the lack of a particular type. ***Function return types, function parameters***, and ***pointers*** are three situations where it is frequently utilized.

```
void processInput(void) { /* Function logic */ }
```