Types of Statements in ‘C’

Types of Statements in C

1. Declaration Statements
2. Input / Output Statements
3. Arithmetic Statements
4. Control Statements

Types of Statements in a Program

- Step 1: START
- Step 2: **DECLARE variables num1, num2 and sum.**
- Step 3: READ values num1 and num2.
- Step 4: COMPUTE sum = num1 + num2
- Step 5: DISPLAY sum
- Step 6: STOP

Set / Block of Statements

Types of Statements in a Program

- Step 1: START
- Step 2: **DECLARE variables num1, num2 and sum.**
- Step 3: READ values num1 and num2.
- Step 4: COMPUTE sum = num1 + num2
- Step 5: DISPLAY sum
- Step 6: STOP

Indicating the **begin** and **end of the Set / Block / program**

Types of Statements in a Program

- Step 1: START
- Step 2: **DECLARE variables num1, num2 and sum.** ← Declaration Statement
- Step 3: READ values num1 and num2.
- Step 4: COMPUTE sum = num1 + num2
- Step 5: DISPLAY sum
- Step 6: STOP

Types of Statements in C

- Statements in ‘C’ are formed using Tokens.

Tokens

1. Variables / Constants
2. Identifiers
3. Keywords
4. String literals
5. Operators
6. Other seperators

Types of Statements in C

1. Declaration Statements
2. Input / Output Statements
3. Arithmetic Statements
4. Control Statements

Declaration Statement

Includes Declaring of:

- Variables
- Constants

Tokens:

1. **Variables / Constants**
2. Identifiers
3. Keywords
4. String literals
5. Operators
6. Other seperators

Variables

- Variables are data that will keep on changing
- A variable is a block of memory that stores data of a particular type and is **named** with an appropriate **identifier**.
- **NAME** of a variable must be a **unique name** that simply references to memory locations, which can hold values (data).

C IDENTIFIERS

- Identifiers give unique names to various objects in a program.
- Are formed by combining letters (both upper and lowercase), digits (0–9) and underscore (_).
- Rules for identifier naming are:
  1. **The first character of an identifier must be a letter (non-digit) including underscore (_).**
  2. **The blank or white space character is not permitted in an identifier. Space, tab, linefeed, carriage-return, formfeed, vertical-tab, and newline characters are "white-space characters“ - they serve the same purpose as the spaces between words and lines on a printed page.**
  3. **Can be any length but implementation dependent (Generally, Maximum 31 Characters).**
  4. Variable names are case sensitive
     - **A** and **a** are different.
  5. **Reserved words/keywords cannot be used.**

C IDENTIFIERS

Examples: variable names

Correct	Wrong
secondName	2ndName /* starts with a digit */
_addNumber	%calculateSum /* contains invalid character */
charAndNum	char /* reserved word */
annual_rate	annual rate /* contains a space */
stage4mark	My\Name /* contains character, \ */

Keywords

- Reserved words in C
- These words are not available for re-definition.
- Have special meaning in C.

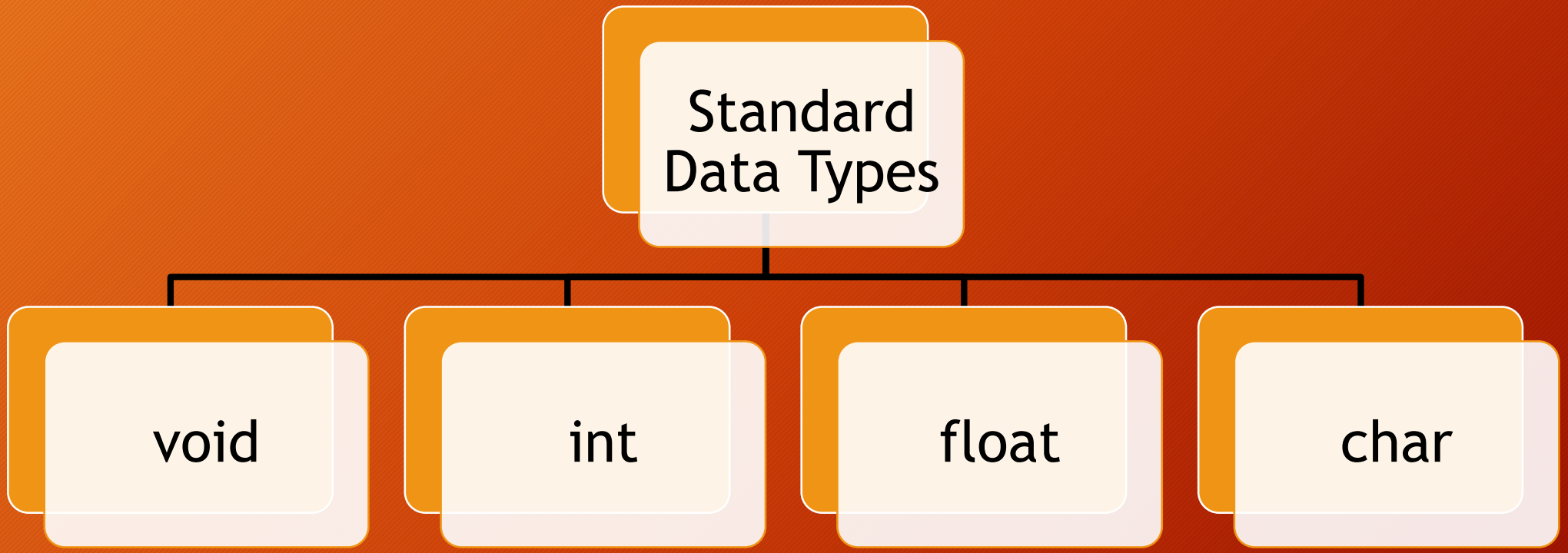
C KEYWORDS/RESERVED WORDS

auto	extern	sizeof	_Alignas (C11)
break	float	static	_Alignof (C11)
case	for	struct	_Atomic (C11)
char	goto	switch	_Bool (C99 beyond)
const	if	typedef	_Complex (C99 beyond)
continue	int	union	_Generic (C11)
default	long	unsigned	_Imaginary (C99 beyond)
do	register	void	_Noreturn (C11)
double	Return	volatile	_Static_assert (C11)
else	short	while	_Thread_local (C11)
enum	signed		inline (C99 beyond)
			restrict (C99 beyond)

Variables

- A variable is a block of memory that stores data of a particular type and is named with an appropriate **identifier**.
- Variables are data that will keep on changing
- *Variables are named memory locations that have a **type**, such as integer or character, which is inherited from their type.*
- *The type determines the values that a variable may contain and the operations that may be used with its values.*

Standard Data Types



Primitive Data Types

Data Types	C-Implementation
void	void
integer	short int (1 or 2 bytes) int (2 or 4 bytes) long int (4 or 8 bytes) unsigned short int (1 or 2 bytes) unsigned int (2 or 4 bytes) unsigned long int (4 or 8 bytes)
character	char (1 byte)
floating point	float (4 bytes) double (8 bytes) long double (10 bytes)

```
/* Program to check the size of data types*/
#include <stdio.h>
main()
{
printf( “%d \n”, sizeof(int));
printf( “%d \n”, sizeof(short int));
printf( “%d \n”, sizeof(long int));
printf( “%d \n”, sizeof(float));
printf( “%d \n”, sizeof(double));
printf( “%d \n”, sizeof(char));
}
```

Primitive Data Types - TASK

- Make a comprehensive notes on each Data type in ‘C’
  - Primitive Data Types
  - Properties
  - Range
  - Size in memory
  - Examples

C VARIABLES - TYPES

- More examples

Correct	Wrong	Comment
`int x, y, z;`	`int 3a, 1, -p;`	
`short number_one;`	`short number+one;`	
`long TypeofCar;`	`long #number`	
`unsigned int positive_number;`	…	
`char Title;`		
`float commission, yield = 4.52;`		
`int my_data = 4;`		
`char the_initial = 'M';`		A char
`char studentName[20] = "Anita";`		**A string**

Declaration and Definition Statement: Variables

1. **Declaration**

   <<Data type>> <<variable name>>;

   int a;

2. **Definition**

   <<varname>>=<<value>>;

   a=10;

- **Operation on variables**

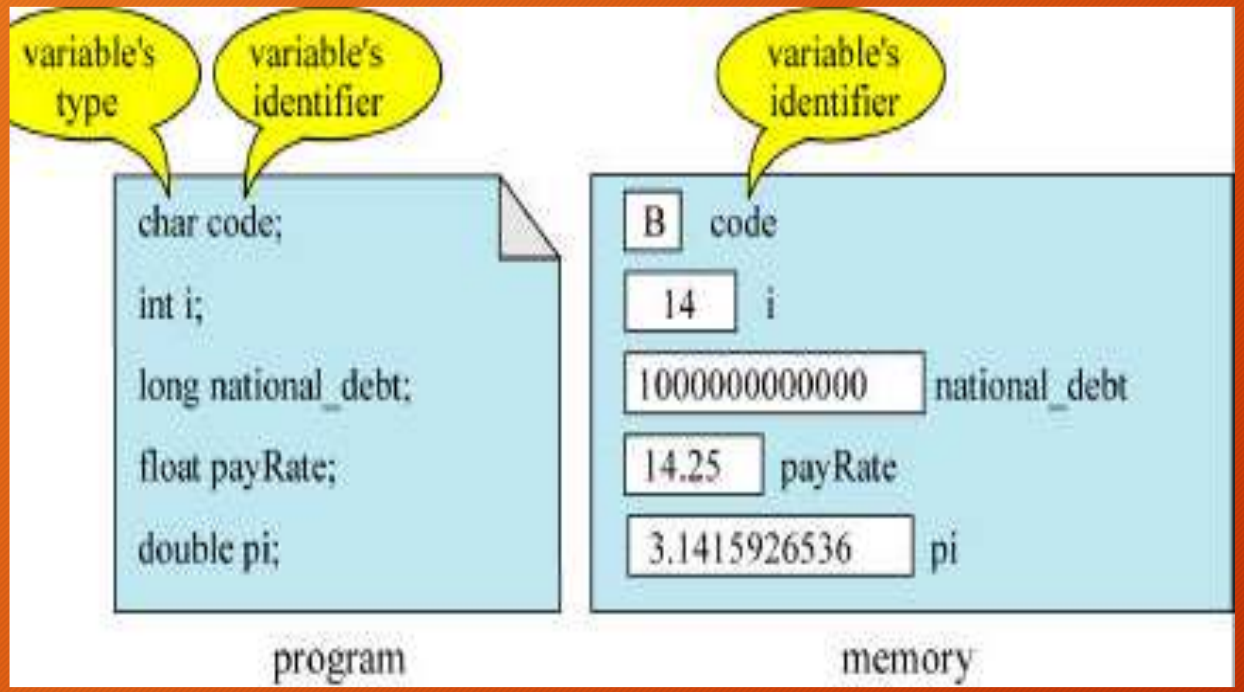
  <<varname>>

  a=a+1; //increments the value of a by 1

Tokens

1. **Variables / Constants**
2. **Identifiers**
3. **Keywords**
4. String literals
5. **Operators**
6. **Other seperators**

Variable Declaration



Variable Initialization

- No variable is initialized until you do so!

```
char code = 'B';
int i = 23456;
long national_debt = 2931000001L;
unsigned long gnp = 332000101LU;
float aVariable = 3.1415f;
double variable2 = 3.1415926535;
long double variable3 = 3.14159265358979L;
```

Variables: TASK

- When the variable is declared for example:
- int i;
- What is the value in i at the time of declatation?