/\*

Operators & expressions of C:

We have following operators:

Assignment Operator

Arithmetic Operators(+,-,\*,/,%)

Logical Operators

Increment/Decrement Operators

Shorthand arithmetic assignment Operators

Conditional Operators

Bitwise Operators

Comma Operator

Some important & commonly used operators Operators:

Assignment Operator

Arithmetic Operators

Logical Operators

Increment/Decrement Operators

Shorthand arithmetic assignment Operators

1.Assignment Operator:

The single equal symbol (=) is refered to as the assigment Operator.

It is used to assign the value of an expression to a variable.

Syntax:

variable=expression;

The expression can be a constant, variable or any valid C expression.

Example:

int a,b,c;

a=10;

b=a;

c=a+b;

2.Arithmetic Operators(+,-,\*,/,%)

#include<stdio.h>

void main()

{

int a=15,b=4;

printf("%d\n",a+b);

printf("%d\n",a-b);

printf("%d\n",a\*b);

printf("%d\n",a/b);

printf("%d\n",a%b);

}

Output:

19

11

60

3

3

Task 1: Perform the above expression after taking input from the user.

Task 2: Evaluate the following expression:

a) a+b\*c

b) a+b/c\*d

Here we can follow BODMAS, however,we can override the default order

of evaluation of an expression by using pair of parenthesis.

Result of a+b\*c is different from (a+b)\*c.

3.Relational Operator:

Relational operators are used to construct relational expressions

which are used to compare two quantities.

For Example:

a>b

a>10

b<a

a==b

a!=b

The value of relational operator is either true(1) or false(0).

For example:

#include<stdio.h>

void main()

{

int a=10,b=15;

printf("%d\n",a>b);

printf("%d\n",b>a);

printf("%d\n",a==b);

printf("%d\n",a!=b);

}

Output:

0

1

0

1

4.Logical Operators:

The logical operators are used to construct compound conditional

expression. We have three logical operators: AND(&&), OR(||) and NOT(!).

The working of AND(&&) operator is depicted below:

Condition 1 Condition 2 Output

True True True

True false False

False True False

False False False

Both/All the conditions must be true to get 'true' as output.

If any condition fails then output will be false.

Note: We can check more than two conditions.

The working of OR(||) operator is depicted below:

Condition 1 Condition 2 Output

True True True

True false true

False True true

False False false

If any of the condition is true then output will be true.

If all the conditions get failed/false, then output will be false.

Note: We can check more than two conditions.

NOT(!) operator:

It reverses the output, from true to false and vice versa.

6.Shorthand Arithmetic Assignment Operators

These are commonly used Shorthand Arithmetic Assignment Operators:

[+=, -==, \*=, /=, %=]

Suppose 'a' is a variable with the value 10. If we want to add 5 to it.

What will we write?

a=a+5;

This assignment statement can now be written as:

a+=5;

It is same as a=a+5;

Similarly:

a=a-5 can be written as a-=5;

a=a\*5 can be written as a\*=5;

a=a/5 can be written as a/=5;

a=a%5 can be written as a%=5;

Example of Shorthand Arithmetic Assignment Operators:

#include<stdio.h>

void main()

{

int a,b,c,d,e;

a=b=c=d=e=10;

a+=5;

b-=5;

c\*=5;

d/=5;

e%=5;

printf("Value of a is=%d\n",a);

printf("Value of b is=%d\n",b);

printf("Value of c is=%d\n",c);

printf("Value of d is=%d\n",d);

printf("Value of e is=%d\n",e);

}

6.Increment/Decrement Operator[++,--]

#include<stdio.h>

void main()

{

int a=6,c=10,b;

b=a++;

printf("Value of b is=%d\n",b);

b=++c;

printf("Value of b is=%d\n",b);

b=c--;

printf("Value of b is=%d\n",b);

b=--c;

printf("Value of b is=%d\n",b);

}

Output:

Value of b is=6

Value of b is=11

Value of b is=11

Value of b is=9

What is type conversion?

It is a conversion from one data type to another data type.

Conversion takes place at two instances.

i) At the time of evaluation of an expression: Whenever an expression has two data items

which are of different types. Lower types converted into higher type. The result of

the expression will be in higher type mode.

The hierarchy of data types in the increasing order is given as follows:

char(The lowest data type)

short

unsigned

long

float

double

long double(The highest data type)

Example:

int a=10;

float f=3.14

a+f=10+3.14

=13.14

ii) At the time of assignment of the value of an expression or source variable

to a target variable. The value of the expression on the right hand side of an

assignment statement gets converted into the type of variable collecting it.

Example:

int c,a=10;

float f=3.14

c=a+f;

Here c is a variable of int type. The right hand side of the assignment statement

gets evaluated to 13.14. But the variable c on the left hand side collects only 13

since it is a variable of int type. If the collecting variable c were of float type,it would

have collected the exact result.

Under some circumstances, automatic type conversion does not work out for us.

For instance-

Consider the arithmetic expression 5/2. In C, the value of this expression would

be 2.

Since both 5 and 2 are integers, the result of the expression would also be

of int type. This is according to the rules of automatic type conversion. But

the exact result of this expression is 2.5.

How can we get this result?

Type casting or forcible conversion is the answer to it.

the general form of casting a value is as follow:

(type-name)expression;

example:

(float)5/2;

Example:

#include<stdio.h>

void main()

{

int a=10,a1;

float f=3.14,f1;

a1=f;

printf("Value of expression is=%d\n",a1);

f1=a;

printf("Value of expression is=%f\n",f1);

a1=5/2;

printf("Value of expression is=%d\n",a1);

a1=5.0/2;

printf("Value of expression is=%d\n",a1);

f1=5/2;

printf("Value of expression is=%f\n",f1);

f1=5.0/2;

printf("Value of expression is=%f\n",f1);

f1=(float)5/2;

printf("Value of expression is=%f\n",f1);

}

Output:

Value of expression is=3

Value of expression is=10.000000

Value of expression is=2

Value of expression is=2

Value of expression is=2.000000

Value of expression is=2.500000

Value of expression is=2.500000

Nested if-else:

Whenever we define if-else block inside another if-else block called

nested if-else.

Syntax:

if (Condition 1)

{

if (Condition 2)

{

Code;

}

else

{

Code;

}

}

else

{

if (Condition 3)

{

Code;

}

else

{

Code;

}

}

Example:

#include<stdio.h>

void main()

{

int a=20,b=10,c=45;

if(a>b)

{

if(a>c)

{

printf("Highest number is:%d \n",a);

}

else

{

printf("Highest number is:%d \n",c);

}

}

else

{

if(b>c)

{

printf("Highest number is:%d \n",b);

}

else

{

printf("Highest number is:%d \n",c);

}

}

}

Else-if Statements:

It is used when we have only one if block, multiple else-if blocks and at the last else block.

Syntax:

if (Condition 1)

{

Statement 1;

}

else if (Condition 2)

{

Statement 2;

}

else

{

Statement 3;

}

Example:

#include<stdio.h>

void main()

{

int num;

printf("Enter any number:\n");

scanf("%d",&num);

if(num>0)

printf("Given number is positive:\n");

else if(num<0)

printf("Given number is negative:\n");

else

printf("Given number is zero:\n");

}

Another example of else-if:

#include<stdio.h>

void main()

{

int num;

printf("Enter any number(0-6):\n");

scanf("%d",&num);

if(num==0)

printf("It is Sunday:\n");

else if(num==1)

printf("It is Monday:\n");

else if(num==2)

printf("It is Tuesday:\n");

else if(num==3)

printf("It is Wednesday:\n");

else if(num==4)

printf("It is Thursday:\n");

else if(num==5)

printf("It is Friday:\n");

else

printf("It is Saturday:\n");

}

Jumps in Loops

break & continue statements

The break Statement

We often come across situations where we want to jump out of a loop instantly,

without waiting to get back to the condition. The keyword 'break' allows us

to do this.

While break is encountered inside a while,for or do-while loop, control

automaticaly passes to the first statement after the loop. A break is

usually associated with an if.

Let us understand the use of break statement.

#include<stdio.h>

void main()

{

int n,i,sum=0,number;

printf("Enter number of elements:\n");

scanf("%d",&n);

for(i=1;i<=n;i++)

{

scanf("%d",&number);

if(number<0)

break;

sum+=number;

}

printf("Sum of number is=%d\n",sum);

}

The continue statement

In some programming situations, we want to take the control to the begining

of the loop, bypassing the statements inside the loop, which have not yet been

executed.The keyword 'continue' allows us to do this.

When continue is encountered inside a loop, control jumps to the begining

of the loop for performing next iteration. A 'continue' is usually associated

with an if.

Let us understand the use of break statement.

#include<stdio.h>

void main()

{

int n,i,sum=0,number;

printf("Enter number of elements:\n");

scanf("%d",&n);

for(i=1;i<=n;i++)

{

scanf("%d",&number);

if(number<0)

continue;

sum+=number;

}

printf("Sum of number is=%d\n",sum);

}

Storage classes

When we declare a variable of some type, the variable name and its data types

are associated with the variable. It is identified by the given variable name

and it is capable of storing value of the specified type.

In addition to these, two important factors are associated with the variable:-

1)Scope of the variable

2)Life time of the variable

Scope of a variable is defined to be the area of its existence in a program.

Lifetime of a variable is defined to be the duration of time for which the

variable exists.

Depending on where variables are declared, they are of two types:

1)Internal variables(Local Variables)

2)External variables(Global Variables)

Internal variables: These are those varaibles which are declared within the

body of a function.Internal variables are known only within the body of their

enclosing functions.

External variables:These are those variables which are declared outside the

functions.They are accessible from the point of declaration till the end of

the program.

Example:

int a;

void main()

{

int b;

}

void myfunction()

{

int c;

}

Here 'a' is a global variable, which is available and accessible for everyone.

But 'b' and 'c' are local variables, they are available and accessible only

for their respective enclosing functions.

Array & Functions:

One dimensional array as arguments to functions.

To make a function take an array of one dimension as its input to be provided

by a calling program, we need to define the function with two

formal parameters:array name with data type specified, and the number of elements

of the array being passed. The function is defined as:

function(array, size)

{

//Your code;

}

or,to be more precise

return-type function\_name(data-type array\_name[], int size)

{

//Your Code;

}

Example:

#include<stdio.h>

void display(int a[], int n)

{

int i;

for(i=0;i<n;i++)

printf("%d\n",a[i]);

}

void main()

{

int i,a[20],n;

printf("Enter number of values:\n");

scanf("%d",&n);

printf("Enter %d values \n",n);

for(i=0;i<n;i++)

scanf("%d",&a[i]);

printf("The elements are \n");

display(a,n);

}

Exercise:

1.WAP to display the elements of 1-D array.

2.WAP to find the sum of elements in a 1-D array.

What is Union(After Structure?

The concept of Union is derived from the concept of structure. The common thing shared

by both structure and union is that both enable us to identify a group of data items

which may be of different types of a common name. But the difference lies in their

storage allocation scheme.

In case of structure, the number of locations allocated would be equal to the number

of members in the structures. Whereas in the case of union, only one loation which is

large enough to collect the largest data type member in the union gets allocated.

This single location can accomodate values of different type at different types

one at a time.

Syntax of union:

union union\_name

{

data-type member 1;

data-type member 2;

data-type member n;

};

As a result of this, a memory allocation gets allocated, the size of which is equal

to that of the largest of the numbers. Accessing the member of a union is similar

to accessing the members of a structure. Dot operator is used to access eash

individual members.

Example:

union Alpha

{

int a;

float b;

char c;

};

union Alpha x;

A variable 'x' is declared to be of type union Alpha. As a result of this, only one

memory location gets allocated. It can be referred to by any one individual member

at any point of time. Note that the size of memory location is 4-bytes, which happens

to be the size of the largest sized data type float in the member list.

Example:

#include<stdio.h>

struct Alpha

{

int a;

float b;

char c;

};

union Beta

{

int a;

float b;

char c;

};

void main()

{

struct Alpha x;

union Beta y;

printf("Size of structure Alpha is=%d\n",sizeof(x));

printf("Size of union Beta is=%d\n",sizeof(y));

}

Output:

Size of structure Alpha is=12

Size of union Beta is=4

Dynamic Memory Allocation:

It is a way to allocate memory to a data structure during runtime.

We need some functions to allocate & free memory dynamically.

Main functions for Dynamic Memory Allocation(DMA):

malloc(): For Memory Allocation

calloc(): For Continuous Allocation

free(): For Freeing of Memory

realloc(): For Reallocation

malloc():

It takes number of bytes to be allocated & returns a pointer of type void.

How to allocate memory:

ptr=(int\*)malloc(n\*sizeof(data-types);

How to check size of different data types:

#include<stdio.h>

void main()

{

printf("%d\n",sizeof(int));

printf("%d\n",sizeof(float));

printf("%d\n",sizeof(char));

}

How to allocate memory:

#include<stdio.h>

#include<stdlib.h>

void main()

{

int i, \*ptr;

ptr=(int\*)malloc(5\*sizeof(int));

ptr[0]=10;

ptr[1]=20;

ptr[2]=30;

ptr[3]=40;

ptr[4]=50;

for(i=0;i<5;i++)

{

printf("%d \n",ptr[i]);

}

}

Practice:

WAP to allocate memory to store 5 prices.

calloc():

It is a continuous allocation which initializes with 0.

ptr=(int\*)calloc(n,sizeof(data-type));

#include<stdio.h>

#include<stdlib.h>

void main()

{

float \*ptr;

int i;

ptr=(float\*)calloc(5,sizeof(float));

for(i=0;i<5;i++)

{

printf("%f\n",ptr[i]);

}

}

WAP to allocate memory of size n, where n is entered by the user.

#include<stdio.h>

#include<stdlib.h>

void main()

{

int \*ptr,i,n;

printf("Enter the value of n:\n");

scanf("%d",&n);

ptr=(int\*)calloc(n,sizeof(int));

for(i=0;i<5;i++)

{

printf("%d\n",ptr[i]);

}

}

free():

We use it to free memory that is allocated using malloc & calloc.

free(ptr);

Example:

#include<stdio.h>

#include<stdlib.h>

void main()

{

int \*ptr;

int n,i;

printf("Enter the value of n: \n");

scanf("%d",&n);

ptr=(int\*)calloc(n,sizeof(int));

for(i=0;i<n;i++)

{

printf("%d\n",ptr[i]);

}

free(ptr);

printf("Now memory is free:\n");

for(i=0;i<n;i++)

{

printf("%d\n",ptr[i]);

}

}

realloc():

realloc(increase or decrease) memory using the same pointer & size.

ptr=realloc(ptr,newsize);

Example:

Allocate memory for 5 numbers, then dynamically increase it to 8 numbers.

#include<stdio.h>

#include<stdlib.h>

void main()

{

int \*ptr,i;

ptr=(int\*)calloc(5,sizeof(int));

printf("Enter number(5):\n");

for(i=0;i<5;i++)

{

scanf("%d",&ptr[i]);

}

ptr=realloc(ptr,8);

printf("Enter number(8):\n");

for(i=0;i<8;i++)

{

scanf("%d",&ptr[i]);

}

printf("Entered 8 values are as follow:\n");

for(i=0;i<8;i++)

{

printf("%d\n",ptr[i]);

}

}

What is File handling?

Storing data in a file and accessing data from a file is called file handling.

What is file pointer?

FILE is a (hidden) structute that needs to be created for opening a file.

A FILE pointer that points to this structure and is used to access the file.

Example:

FILE fp;

Different modes of File.

r : For Reading

w : For Writing

a : For Appending

rb : Open to read in binary

wb : Open to write in binary

Some important file handling operations.

Create

Read

Write

Delete

Copy

How to create a file?

#include<stdio.h>

void main()

{

FILE \*fp;

fp=fopen("Alpha.txt","w");

printf("File is created successfully:\n");

fclose(fp);

}

How to write in the file?

#include<stdio.h>

void main()

{

FILE \*fp;

int num;

fp=fopen("Alpha.txt","w");

if(fp==NULL)

{

printf("File is not created!\n");

}

printf("File created successfully:\n");

printf("Enter a number:\n");

scanf("%d",&num);

printf("Data has been written successfully:\n");

fprintf(fp,"%d",num);

fclose(fp);

}

How to read from the file?

#include<stdio.h>

void main()

{

FILE \*fp;

int num;

if((fp=fopen("Alpha.txt","r"))==NULL)

{

printf("Error Occurred:\n");

exit(1);

}

fscanf(fp,"%d",&num);

printf("%d\n",num);

fclose(fp);

}

Additional Program for writing and reading a file.

How to write in the file?

void main()

{

FILE \*fp;

char c;

fp=fopen("Beta.txt","w");

printf("Please write in the file:\n");

while((c=getchar())!=EOF)

putc(c,fp);

fclose(fp);

}

How to read a file?

#include<stdio.h>

void main()

{

FILE \*fp;

char c;

fp=fopen("Beta.txt","r");

if(fp==NULL)

{

printf("File not found!\n");

}

while((c=getc(fp))!=EOF)

printf("%c",c);

fclose(fp);

}

\*/