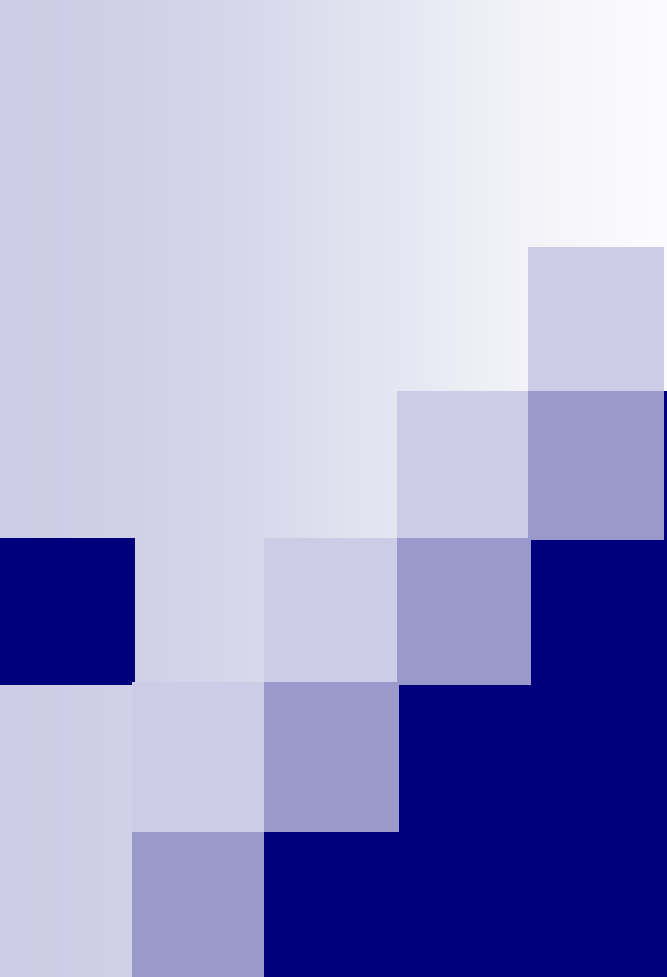




Conditional Statements

Statements in a C program

- Parts of C program that tell the computer what to do
- Different types
 - Declaration statements
 - Declares variables etc.
 - Assignment statement
 - Assignment expression, followed by a ;
 - Control statements
 - For branching and looping, like if-else, for, while, do-while (to be seen later)
 - Input/Output
 - Read/print, like printf/scanf

Example

Declaration statement

```
int a, b, larger;
```

Input/Output statement

```
scanf("%d %d", &a, &b);
```

Assignment statement

```
larger = b;
```

Control statement

```
if (a > b)
```

Assignment statement

```
    larger = a;
```

Input/Output statement

```
printf("Larger number is %d\n", larger);
```



■ Compound statements

- A sequence of statements enclosed within { and }
- We will also call it block of statements informally
- Each statement in a block can be an assignment statement, control statement, input/output statement, or another compound statement
- There may be only one statement inside a block also

Example

```
int n;  
scanf("%d", &n);  
while(1) {  
    if (n > 0) break;  
    scanf("%d", &n);  
}
```

Compound statement

Conditional Statements

- Allow different sets of instructions to be executed depending on truth or falsity of a logical condition
- Also called **Branching**
- How do we specify conditions?
 - Using expressions
 - non-zero value means condition is **true**
 - value 0 means condition is **false**
 - Usually logical expressions, but can be any expression
 - The value of the expression will be used

Branching: **if** Statement

```
if (expression)  
    statement;
```

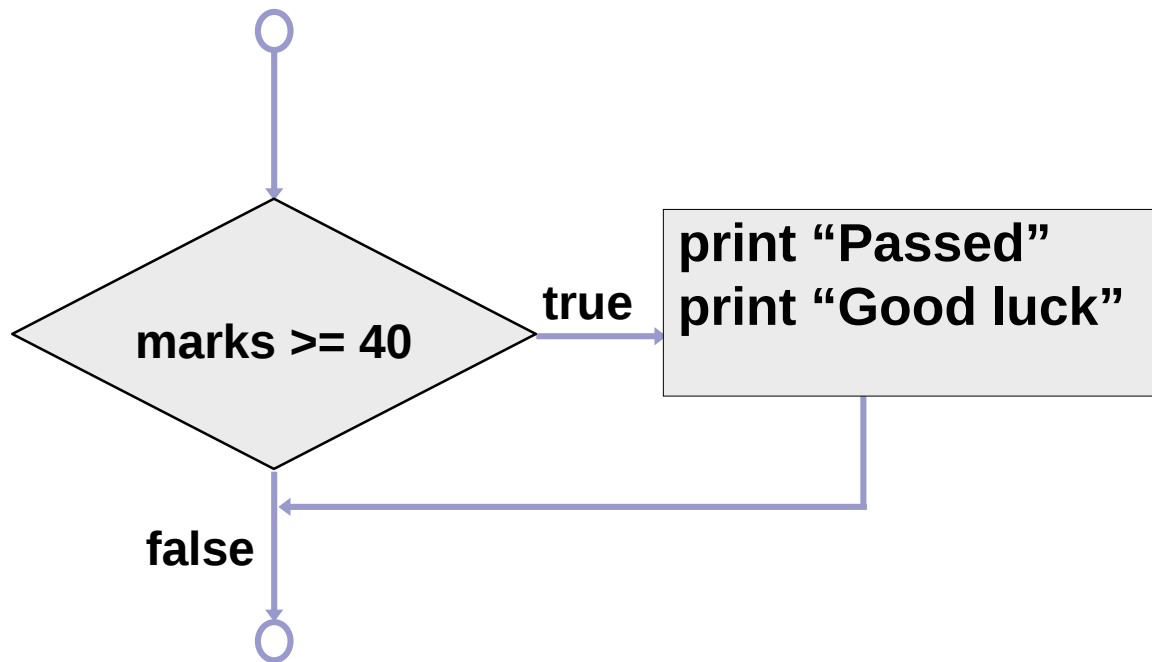
```
if (expression) {  
    Block of statements;  
}
```

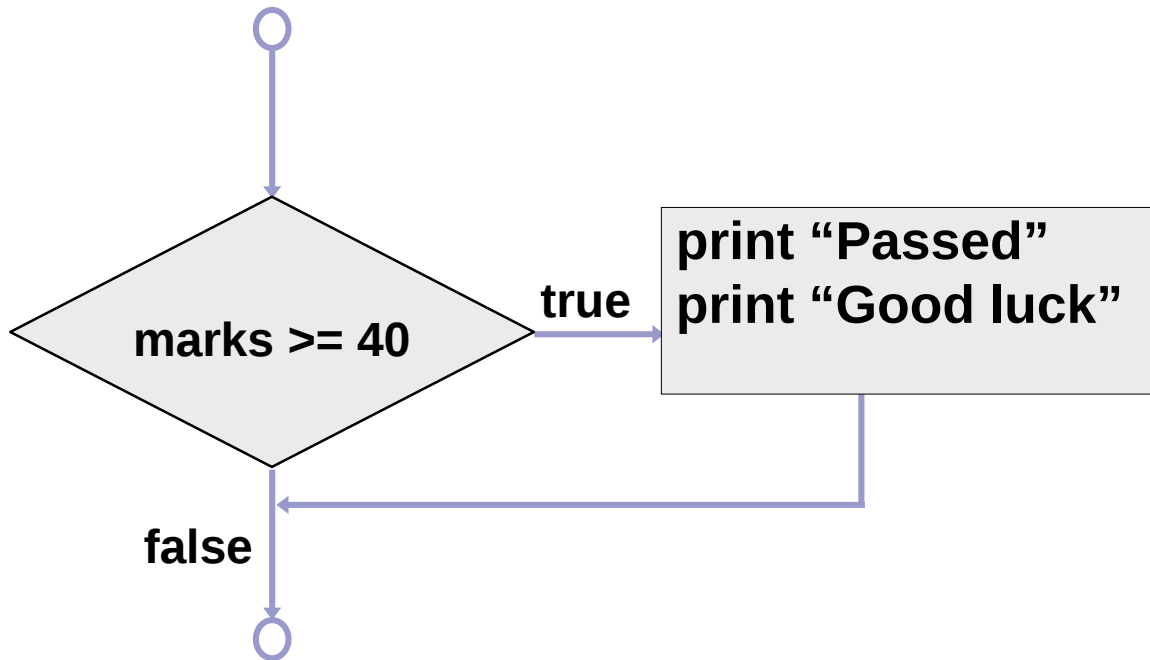
Branching: **if** Statement

```
if (expression)  
    statement;
```

```
if (expression) {  
    Block of statements;  
}
```

The condition to be tested is any expression enclosed in parentheses. The expression is evaluated, and if its value is non-zero, the statement/block of statements is executed.

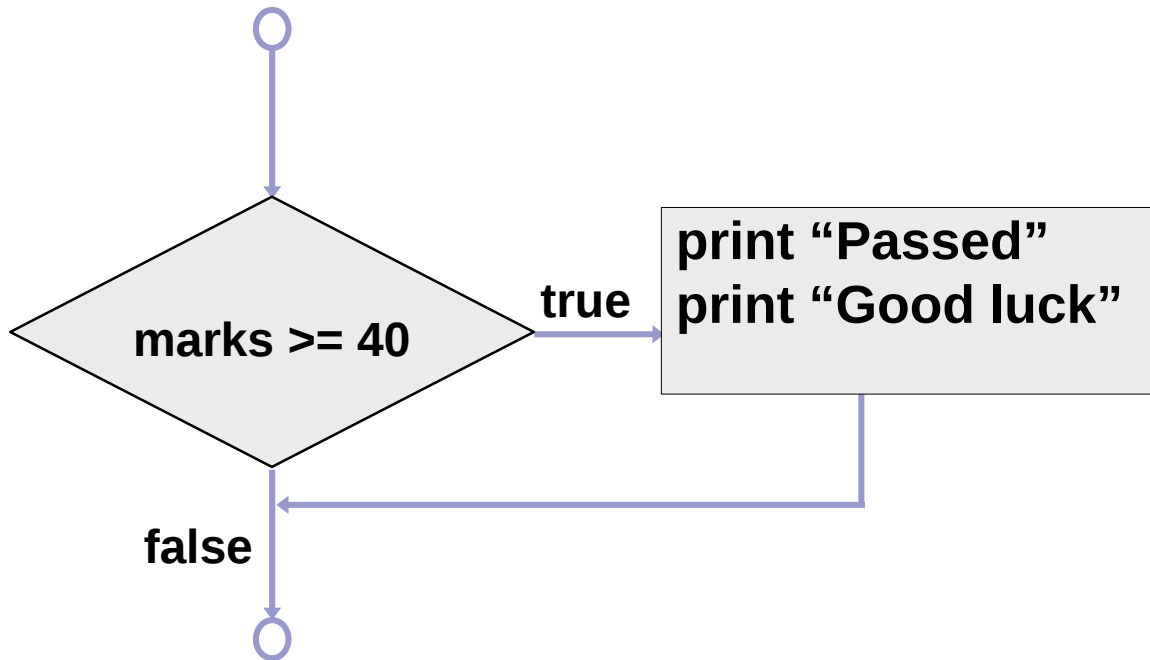




A decision can be made on any expression.

zero - false

nonzero - true



A decision can be made on any expression.

zero - false

nonzero - true

```
if (marks >= 40) {  
    printf("Passed \n");  
    printf("Good luck\n");  
}  
printf ("End\n") ;
```

Branching: **if-else** Statement

```
if (expression) {  
    Block of  
    statements;  
}  
else {  
    Block of  
    statements;  
}
```

```
if (expression) {  
    Block of statements;  
}  
else if (expression) {  
    Block of statements;  
}  
else {  
    Block of statements;  
}
```

Grade Computation

```
int main() {  
    int marks;  
    scanf("%d", &marks);  
    if (marks >= 80)  
        printf ("A") ;  
    else if (marks >= 70)  
        printf ("B") ;  
    else if (marks >= 60)  
        printf ("C") ;  
    else printf ("Failed");  
    return 0;  
}  
  
if (marks >= 80)  
    printf ("A") ;  
else {  
    if (marks >= 70)  
        printf ("B") ;  
    else {  
        if (marks >= 60)  
            printf ("C") ;  
        else printf ("Failed");  
    }  
}
```

```
int main () {  
    int marks;  
    scanf ("%d", &marks) ;  
    if (marks >= 80) {  
        printf ("A: ") ;  
        printf ("Good Job!") ;  
    }  
    else if (marks >= 70) printf ("B ") ;  
    else if (marks >= 60) printf ("C ") ;  
    else {  
        printf ("Failed: ") ;  
        printf ("Study hard!") ;  
    }  
    return 0;  
}
```

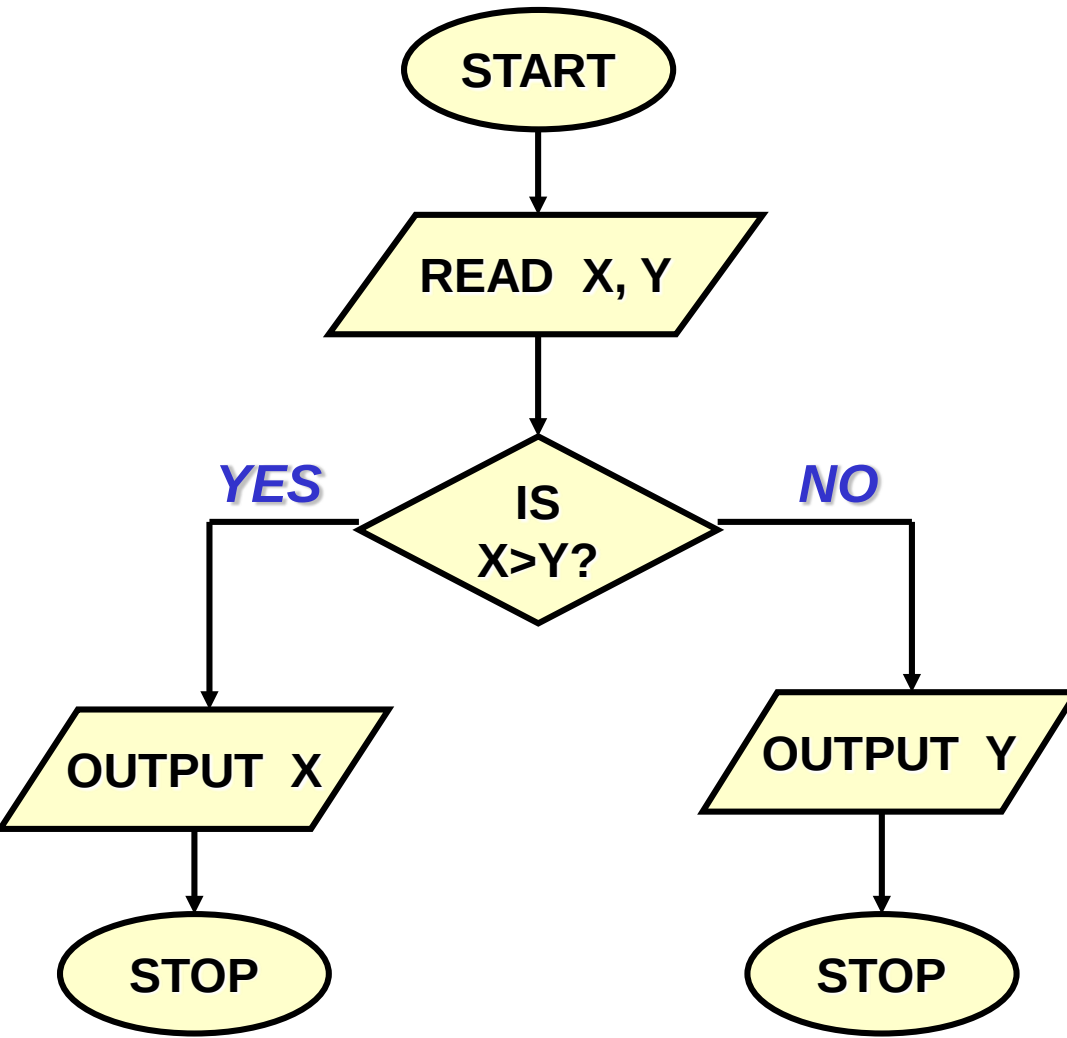
Outputs for different inputs

90
A: Good Job!

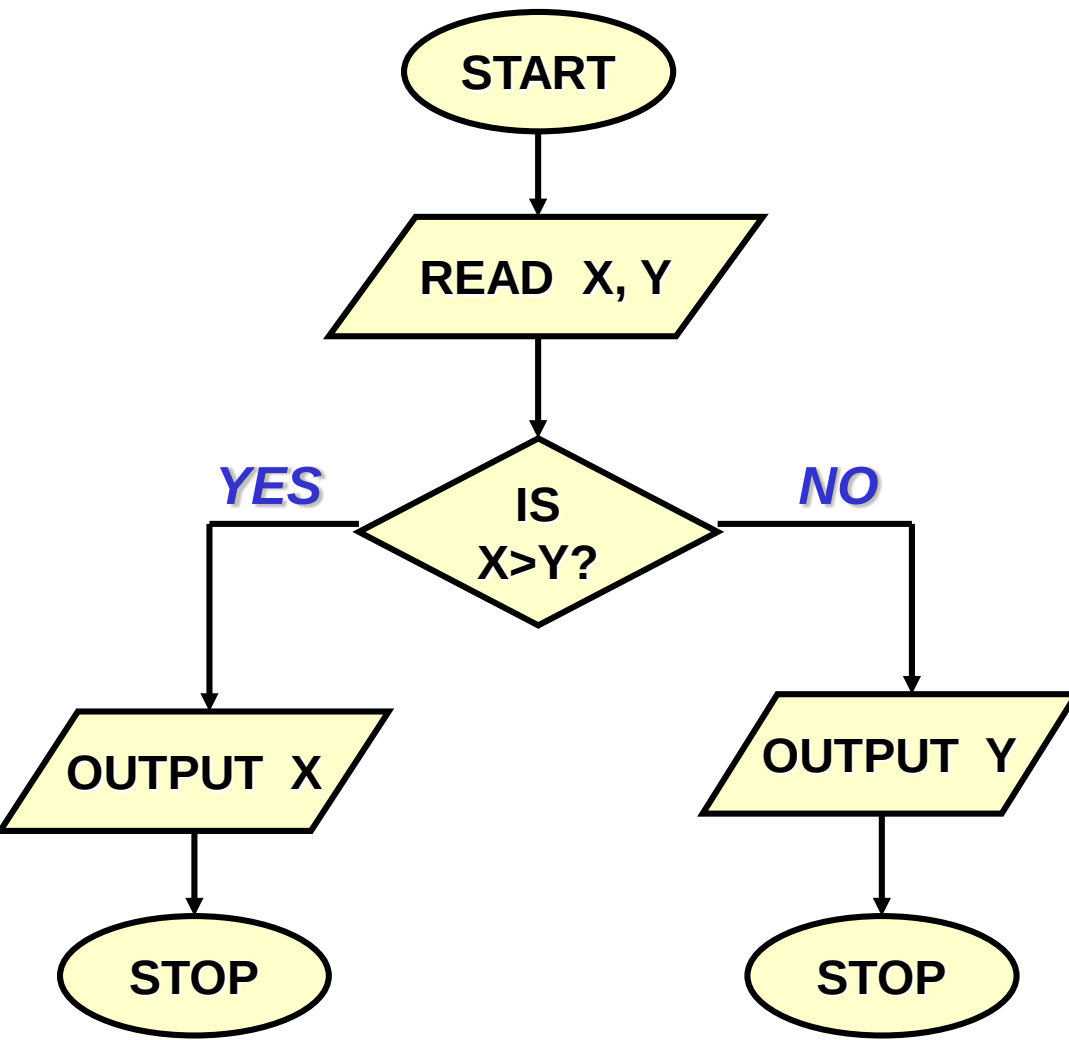
65
C

50
Failed: Study hard!

Find the larger of two numbers

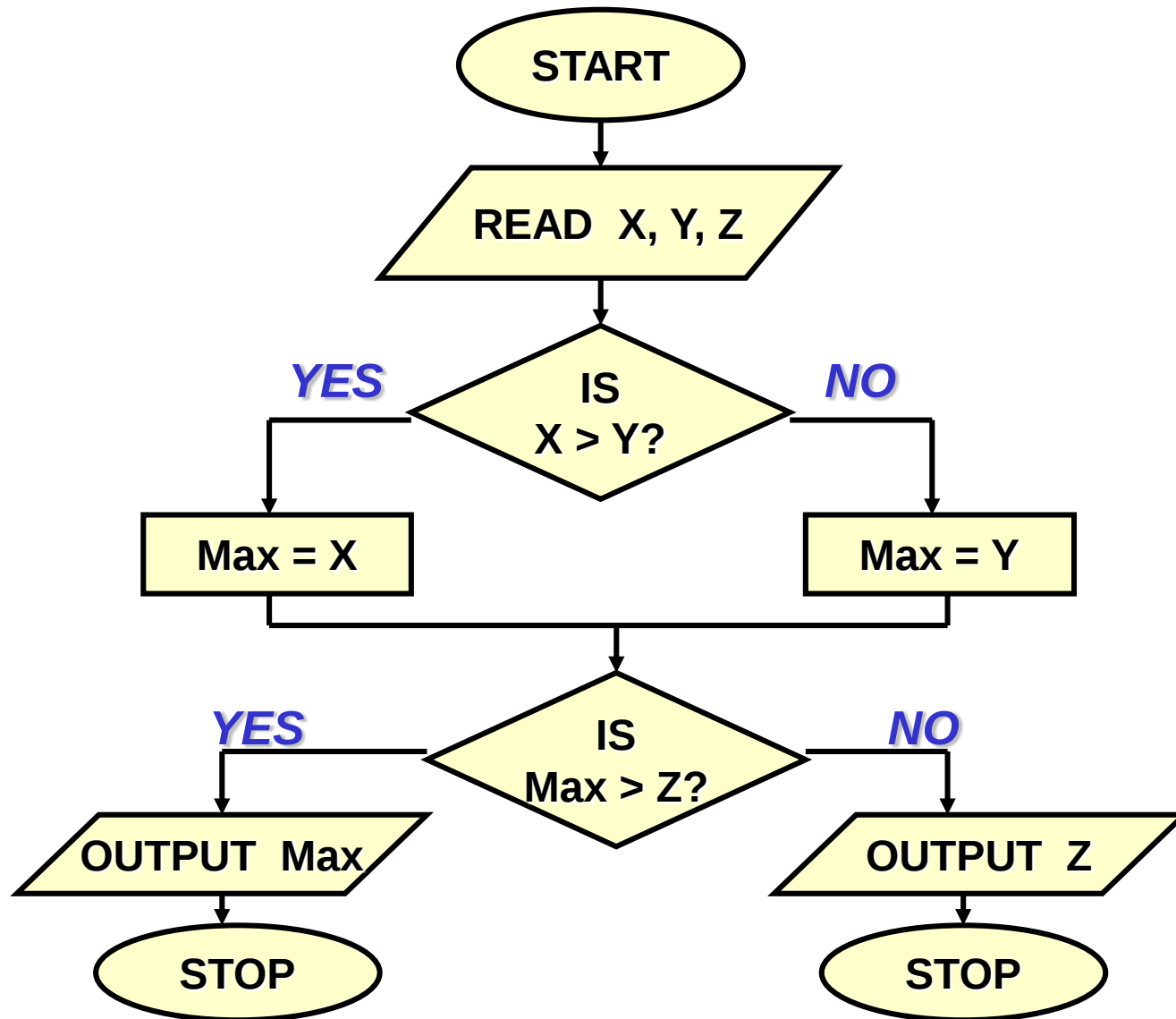


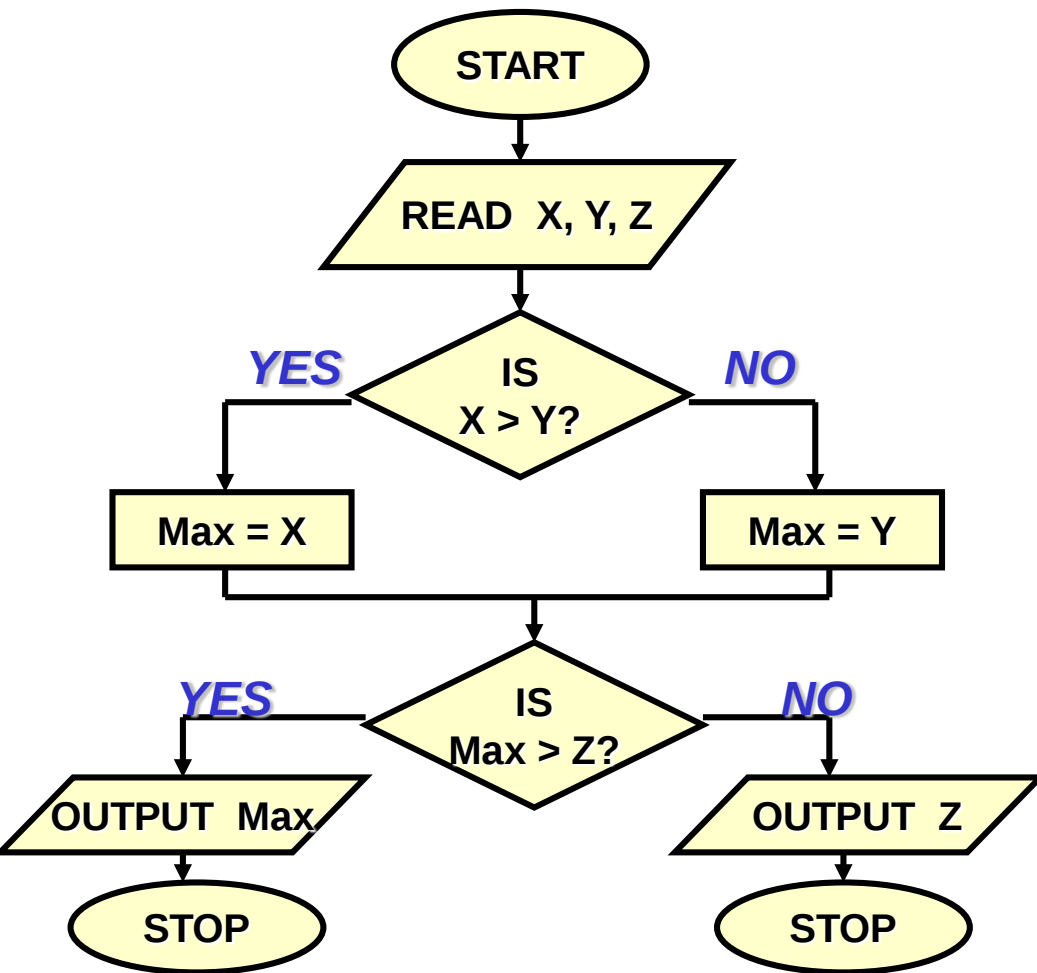
Find the larger of two numbers

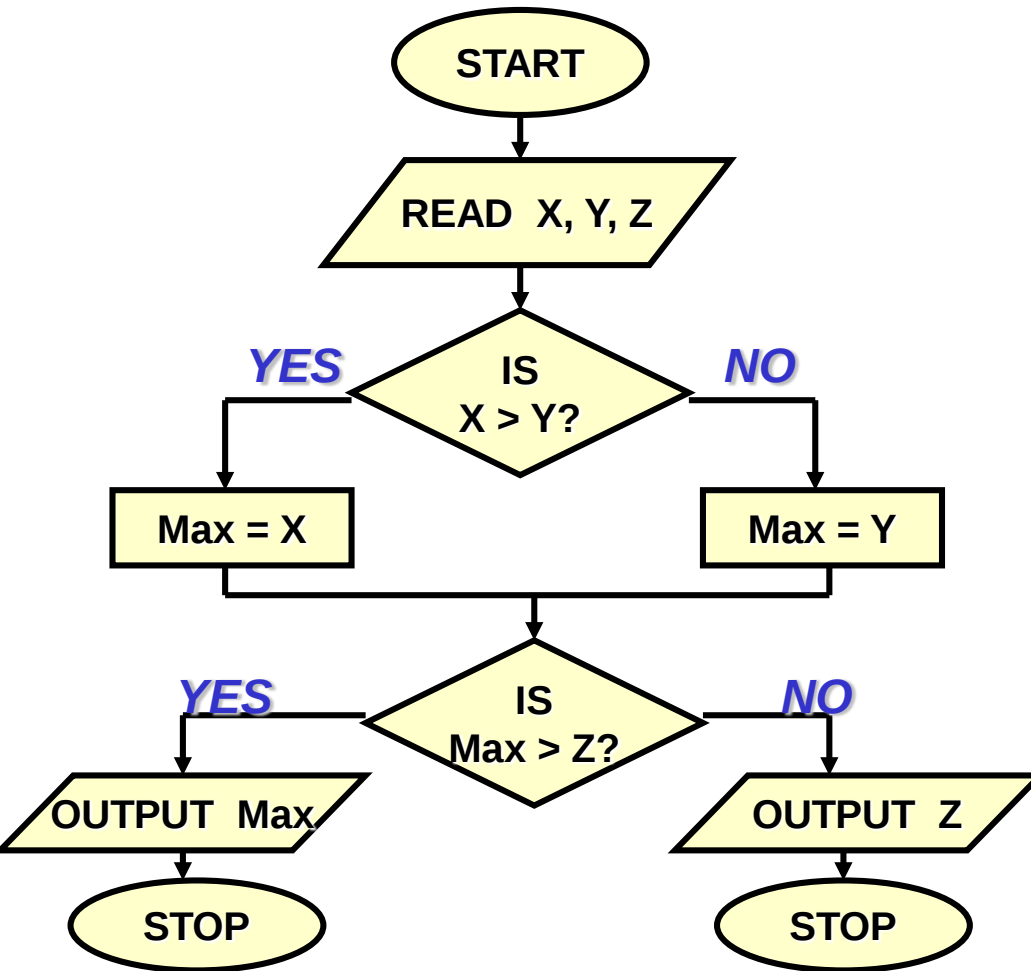


```
int main () {  
    int x, y;  
    scanf ("%d%d", &x, &y);  
    if (x > y)  
        printf ("%d\n", x);  
    else  
        printf ("%d\n", y);  
    return 0;  
}
```

Largest of three numbers







```
int main () {  
    int x, y, z, max;  
    scanf ("%d%d%d",&x,&y,&z);  
    if (x > y)  
        max = x;  
    else max = y;  
    if (max > z)  
        printf ("%d", max) ;  
    else printf ("%d",z);  
    return 0;  
}
```

Another version

```
int main() {  
    int a,b,c;  
    scanf ("%d%d%d", &a, &b, &c);  
    if ((a >= b) && (a >= c))  
        printf ("\n The largest number is: %d", a);  
    if ((b >= a) && (b >= c))  
        printf ("\n The largest number is: %d", b);  
    if ((c >= a) && (c >= b))  
        printf ("\n The largest number is: %d", c);  
    return 0;  
}
```

Confusing Equality (==) and Assignment (=) Operators

■ Dangerous error

- Does not ordinarily cause syntax errors
- Any expression that produces a value can be used in control structures
- Nonzero values are true, zero values are false

■ Example:

```
if ( payCode = 4 )  
    printf( "You get a bonus!\n" );
```

WRONG! Will always print the line

Nesting of if-else Structures

- It is possible to nest if-else statements, one within another
- All “if” statements may not be having the “else” part
 - Confusion??
- Rule to be remembered
 - An “else” clause is associated with the closest preceding unmatched “if”

Dangling else problem

if (exp1) if (exp2) stmta else stmtb

```
if (exp1) {  
    if (exp2)  
        stmta  
    else  
        stmtb  
}
```

OR

```
if (exp1) {  
    if (exp2)  
        stmta  
}  
else  
    stmtb
```

?

Which one is the correct interpretation?

Give braces explicitly in your programs to match the else with the correct if to remove any ambiguity

More Examples

```
if e1 s1  
else if e2 s2
```

```
if e1 s1  
else if e2 s2  
else s3
```

?

```
if e1 if e2 s1  
else s2  
else s3
```

Answers

if e1 s1
else if e2 s2



if e1 s1
else { if e2 s2 }

if e1 s1
else if e2 s2
else s3



if e1 s1
else { if e2 s2
 else s3 }

if e1 if e2 s1
else s2
else s3



if e1 { if e2 s1
 else s2 }
else s3

While programming, it is always good to explicitly give the { and } to avoid any mistakes

Example

```
int main()
{
    int x;
    scanf("%d", &x);
    if (x >= 0)
        if (x <= 100)
            printf("ABC\n");
    else
        printf("XYZ\n");
    return 0;
}
```

Print "ABC" if a number is between 0 and 100, or "XYZ" if it is -ve. Do not print anything in other cases.

Example

```
int main()
{
    int x;
    scanf("%d", &x);
    if (x >= 0)
        if (x <= 100)
            printf("ABC\n");
    else
        printf("XYZ\n");
    return 0;
}
```

Print "ABC" if a number is between 0 and 100, or "XYZ" if it is -ve. Do not print anything in other cases.

Outputs for different inputs

150
XYZ

Not what we want, should not have printed anything

-20

Not what we want, should have printed XYZ

Correct Program

```
int main()
{
    int x;
    scanf("%d", &x);
    if (x >= 0)
    {
        if (x <= 100)
            printf("ABC\n");
    }
    else
        printf("XYZ\n");
    return 0;
}
```

Outputs for different inputs

150

-20
XYZ

The Conditional Operator ?:

- This makes use of an expression that is either non-0 or 0. An appropriate value is selected, depending on the value of the expression

- Example: instead of writing

if (balance > 5000)

interest = balance * 0.2;

else interest = balance * 0.1;

We can just write

interest = (balance > 5000) ? balance * 0.2 :
balance * 0.1;

More Examples

- ```
if (((a > 10) && (b < 5)))
 x = a + b;
else x = 0;
```

$x = ((a > 10) \ \&\& \ (b < 5)) ? a + b : 0$

- ```
if (marks >= 60)  
    printf("Passed \n");  
else printf("Failed \n");
```

$(marks \geq 60) ? \text{printf}(\text{"Passed \n"}) : \text{printf}(\text{"Failed \n"})$;

The **switch** Statement

- An alternative to writing lots of if-else in some special cases
- This causes a particular group of statements to be chosen from several available groups based on equality tests only
- Uses **switch** statement and **case** labels

■ Syntax

```
switch (expression) {  
    case const-expr-1: S-1  
    case const-expr-2: S-2  
    :  
    case const-expr-m: S-m  
    default: S  
}
```

- **expression** is any integer-valued expression
- **const-expr-1, const-expr-2,...** are any **constant** integer-valued expressions
 - Values must be distinct
- **S-1, S-2, ..., S-m, S** are statements/compound statements
- Default is optional, and can come anywhere (not necessarily at the end as shown)

Behavior of `switch`

- `expression` is first evaluated
- It is then compared with `const-expr-1`, `const-expr-2`,...for equality **in order**
- If it matches any one, **all statements from that point till the end of the switch are executed** (including statements for default, if present)
 - Use `break` statements if you do not want this (see example)
- Statements corresponding to `default`, if present, are executed if no other expression matches

Example

```
int main()
{
    int x;
    scanf("%d", &x);
    switch (x) {
        case 1: printf("One\n");
        case 2: printf("Two\n");
        default: printf("Not one or two\n");
    };
}
```

If x = 1 is entered, this will print

One

Two

Not one or two

Not what we want

switch-1.c

Correct Program

```
int main()
{
    int x;
    scanf("%d", &x);
    switch (x) {
        case 1: printf("One\n");
                break;
        case 2: printf("Two\n");
                break;
        default: printf("Not one or two\n");
    };
}
```

If x = 1 is entered, this will print

One

Rounding a Digit

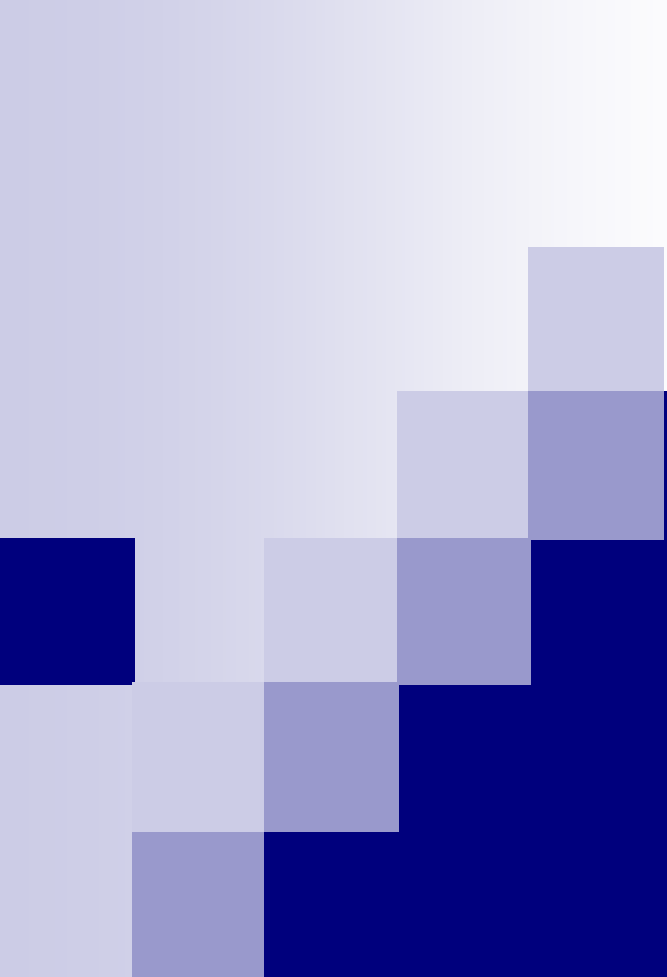
```
switch (digit) {  
    case 0:  
    case 1:  
    case 2:  
    case 3:  
    case 4: result = 0; printf ("Round down\n"); break;  
    case 5:  
    case 6:  
    case 7:  
    case 8:  
    case 9: result = 10; printf("Round up\n"); break;  
}
```

Since there isn't a break statement here, the control passes to the next statement without checking the next condition.

It will come here if digit is any of 0 to 4. Round to 0, then break as done.

The **break** Statement

- Used to exit from a switch or terminate from a loop
- With respect to “switch”, the “break” statement causes a transfer of control out of the entire “switch” statement, to the first statement following the “switch” statement
- Can be used with other statements also ... (will show later)



More on Data Types

More Data Types in C

- Some of the basic data types can be augmented by using certain data type qualifiers:
 - ☐ short
 - ☐ long
 - ☐ signed
 - ☐ unsigned

size qualifier

sign qualifier
- Typical examples:
 - ☐ short int (usually 2 bytes)
 - ☐ long int (usually 4 bytes)
 - ☐ unsigned int (usually 4 bytes, but no way to store + or -)

Some typical sizes (some of these can vary depending on type of machine)

Integer data type	#Bits	Minimum value	Maximum value
char	8	$-2^7 = -128$	$2^7 - 1 = 127$
short int	16	$-2^{15} = -32768$	$2^{15} - 1 = 32767$
int	32	$-2^{31} = -2147483648$	$2^{31} - 1 = 2147483647$
long int	32	$-2^{31} = -2147483648$	$2^{31} - 1 = 2147483647$
long long int	64	$-2^{63} = -9223372036854775808$	$2^{63} - 1 = 9223372036854775807$
unsigned char	8	0	$2^8 - 1 = 255$
unsigned short int	16	0	$2^{16} - 1 = 65535$
unsigned int	32	0	$2^{32} - 1 = 4294967295$
unsigned long int	32	0	$2^{32} - 1 = 4294967295$
unsigned long long int	64	0	$2^{64} - 1 = 18446744073709551615$

More on the `char` type

- Is actually stored as an integer internally
- Each character has an integer code associated with it (`ASCII` code value)
- Internally, storing a character means storing its integer code
- All operators allowed on `int` are also allowed on `char`
 - $32 + \text{'a'}$ will evaluate to $32 + 97$ (the integer `ascii` code of the character `'a'`) = 129
 - Same for other operators
- Can switch on chars constants in `switch`, as they are integer constants

Another example

```
int a;  
a = 'c' * 3 + 5;  
printf("%d", a);
```

**Will print 302 ($99*3 + 5$)
(ASCII code of 'c' = 99)**

```
char c = 'A';  
printf("%c = %d", c, c);
```

**Will print A = 65
(ASCII code of 'A' = 65)**

Assigning char to int is fine. But other way round is dangerous, as size of int is larger

ASCII Code

- Each character is assigned a unique integer value (code) between 32 and 127
- The code of a character is represented by an 8-bit unit. Since an 8-bit unit can hold a total of $2^8=256$ values and the computer character set is much smaller than that, some values of this 8-bit unit do not correspond to visible characters
- But never try to remember exact ASCII codes while programming. Use the facts that
 - C stores characters as integers
 - Ascii codes of some important characters are contiguous (digits, lowercase alphabets, uppercase alphabets)

Decimal	Hex	Binary	Character	Decimal	Hex	Binary	Character
32	20	00100000	SPACE	80	50	01010000	P
33	21	00100001	!	81	51	01010001	Q
34	22	00100010	"	82	52	01010010	R
35	23	00100011	#	83	53	01010011	S
36	24	00100100	\$	84	54	01010100	T
37	25	00100101	%	85	55	01010101	U
38	26	00100110	&	86	56	01010110	V
39	27	00100111	'	87	57	01010111	W
40	28	00101000	(	88	58	01011000	X
41	29	00101001	)	89	59	01011001	Y
42	2a	00101010	*	90	5a	01011010	Z
43	2b	00101011	+	91	5b	01011011	[
44	2c	00101100	,	92	5c	01011100	\
45	2d	00101101	-	93	5d	01011101	]
46	2e	00101110	.	94	5e	01011110	^
47	2f	00101111	/	95	5f	01011111	_
48	30	00110000	0	96	60	01100000	`
49	31	00110001	1	97	61	01100001	a
50	32	00110010	2	98	62	01100010	b

51	33	00110011	3	99	63	01100011	c
52	34	00110100	4	100	64	01100100	d
53	35	00110101	5	101	65	01100101	e
54	36	00110110	6	102	66	01100110	f
55	37	00110111	7	103	67	01100111	g
56	38	00111000	8	104	68	01101000	h
57	39	00111001	9	105	69	01101001	i
58	3a	00111010	:	106	6a	01101010	j
59	3b	00111011	;	107	6b	01101011	k
60	3c	00111100	<	108	6c	01101100	l
61	3d	00111101	=	109	6d	01101101	m
62	3e	00111110	>	110	6e	01101110	n
63	3f	00111111	?	111	6f	01101111	o
64	40	01000000	@	112	70	01110000	p
65	41	01000001	A	113	71	01110001	q
66	42	01000010	B	114	72	01110010	r
67	43	01000011	C	115	73	01110011	s
68	44	01000100	D	116	74	01110100	t
69	45	01000101	E	117	75	01110101	u
70	46	01000110	F	118	76	01110110	v

71	47	01000111	G		119	77	01110111	w
72	48	01001000	H		120	78	01111000	x
73	49	01001001	I		121	79	01111001	y
74	4a	01001010	J		122	7a	01111010	z
75	4b	01001011	K		123	7b	01111011	{
76	4c	01001100	L		124	7c	01111100	
77	4d	01001101	M		125	7d	01111101	}
78	4e	01001110	N		126	7e	01111110	~
79	4f	01001111	O		127	7f	01111111	DELETE

Example: checking if a character is a lowercase alphabet

```
int main()
{
    char c1;
    scanf("%c", &c1);
    /* the ascii code of c1 must lie between the
       ascii codes of 'a' and 'z' */
    if (c1 >= 'a' && c1 <= 'z')
        printf("%c is a lowercase alphabet\n", c1);
    else printf("%c is not a lowercase alphabet\n", c1);
    return 0;
}
```

Example: converting a character from lowercase to uppercase

```
int main()
{
    char c1;
    scanf("%c", &c1);
    /* convert to uppercase if lowercase, else leave as it is */
    if (c1 >= 'a' && c1 <= 'z')
        /* since ascii codes of uppercase letters are contiguous, the
           uppercase version of c1 will be as far away from the ascii code
           of 'A' as it is from the ascii code of 'a' */
        c1 = 'A' + (c1 - 'a');
    printf(("The letter is %c\n", c1);
    return 0;
}
```

Switching with char type

```
char letter;  
scanf("%c", &letter);  
switch ( letter ) {  
    case 'A':  
        printf ("First letter \n");  
        break;  
    case 'Z':  
        printf ("Last letter \n");  
        break;  
    default :  
        printf ("Middle letter \n");  
}
```

Switching with char type

```
char letter;  
scanf("%c", &letter);  
switch ( letter ) {  
    case 'A':  
        printf ("First letter \n");  
        break;  
    case 'Z':  
        printf ("Last letter \n");  
        break;  
    default :  
        printf ("Middle letter \n");  
}
```

Will print this statement
for all letters other than
A or Z

Another Example

```
switch (choice = getchar()) {  
    case 'r' :  
        case 'R': printf("Red");  
                    break;  
    case 'b' :  
        case 'B': printf("Blue");  
                    break;  
    case 'g' :  
        case 'G': printf("Green");  
                    break;  
    default: printf("Black");  
}
```

Another Example

```
switch (choice = getchar()) {  
    case 'r' :  
    case 'R': printf("Red");  
               break;  
  
    case 'b' :  
    case 'B': printf("Blue");  
               break;  
  
    case 'g' :  
    case 'G': printf("Green");  
               break;  
  
    default: printf("Black");  
}
```

Since there isn't a break statement here, the control passes to the next statement (printf) without checking the next condition.

Evaluating expressions

```
int main () {  
    int operand1, operand2;  
    int result = 0;  
    char operation ;  
    /* Get the input values */  
    printf ("Enter operand1 :");  
    scanf ("%d",&operand1) ;  
    printf ("Enter operation :");  
    scanf ("\n%c",&operation);  
    printf ("Enter operand 2 :");  
    scanf ("%d", &operand2);  
    switch (operation) {  
        case '+':  
            result=operand1+operand2;  
            break;
```

```
        case '-':  
            result=operand1-operand2;  
            break;  
        case '*':  
            result=operand1*operand2;  
            break;  
        case '/':  
            if (operand2 !=0)  
                result=operand1/operand2;  
            else  
                printf("Divide by 0 error");  
            break;  
        default:  
            printf("Invalid operation\n");  
            return;  
    }  
    printf ("The answer is %d\n",result);  
    return 0;  
}
```

Practice Problems

1. Read in 3 integers and print a message if any one of them is equal to the sum of the other two.
2. Read in the coordinates of two points and print the equation of the line joining them in $y = mx + c$ form.
3. Read in the coordinates of 3 points in 2-d plane and check if they are collinear. Print a suitable message.
4. Read in the coordinates of a point, and the center and radius of a circle. Check and print if the point is inside or outside the circle.
5. Read in the coefficients a , b , c of the quadratic equation $ax^2 + bx + c = 0$, and print its roots nicely (for imaginary roots, print in $x + iy$ form)
6. Suppose the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 are mapped to the lowercase letters a, b, c, d, e, f, g, h, i, j respectively. Read in a single digit integer as a character (using `%c` in `scanf`) and print its corresponding lowercase letter. Do this both using switch and without using switch (two programs). Do not use any ascii code value directly.
7. Suppose that you have to print the grades of a student, with ≥ 90 marks getting EX, 80-89 getting A, 70-79 getting B, 60-69 getting C, 50-59 getting D, 35-49 getting P and < 30 getting F. Read in the marks of a student and print his/her grade.