About your final

- 2 hour long closed-book exam.
- May carry a single-sided A4 cheat sheet of your own.
- You may write anything but examples of source code (pieces >=5 statements).
- Your cheat sheet must be submitted together with the exam paper after the exam.

About your final

- 35 multiple choice questions (35%)
- 15 true-or-false questions (15%)
- 2 program-completion questions (10%)
- 4 programing questions (40%)

  (to write complete programs/functions)

C language programing

Structure of C Program

	1	`#include <stdio.h>`	Header
	2	`int main(void)`	Main
BODY	3	`{`	
	4	`printf("Hello World");`	Statement
	5	`return 0;`	Return
	6	`}`	

Head files (for interface of library functions) and macro definitions

```
#define Pi 3.14159265
```

Processed by pre-compiler

Identifiers in C: can include letters (a-z or A-Z, ***case sensitive***), digits (0-9), and underscore (_), and must begin with letters or underscore.

Main function

C programs consist of functions, the “main” function is the entry of the entire program

```
int main(void)
{
    body statements;
    return 0;
}
```

```
int main(int argc, char *argv[])
{
    body statements;
    return 0;
}
```

Processing additional arguments (from the command line) as an array of char pointers (const strings)

Functions

```
double average(double a, double b);

double average(double a, double b)
{
    double c = (a + b) / 2;
    return c;
}

int main(void)
{
    double avg, a=0.5, b=2.0;
    avg = average(a, b);
    return 0;
}
```

Return type

Parameters (dummy, 形参)

Must return except for a void function

Name of the function (must match)

Arguments (real, 实参)

Calling

Declaration/interface

Before calling, may be in a header file and included with #include

Definition/implementation

May be placed anywhere, even in a separate file.

Variables

All variables must be declared before usage!

```
int Global_var1, Global_var2;

double average(double a, double b)
{
    double c = (a + b) / 2;
    return c;
}

int main(void)
{
    double avg, a=0.5, b=2.0;
    avg = average(a, b);
    {
        int i;…
    }
    …
}
```

Global variables
Can be accessed everywhere

Local variables
Can only be accessed locally

accessible

accessible

inaccessible

Data types

modifiers		const.	size (bytes)
	char	`a`	1
	short (**int**)		2
(**unsigned**)	**int**	-1234 1234u	4
	long (**int**)	1234L	4
	long long (**int**)	1234LL 1234ULL	8
			16

		const. lit.
	float	-1.234f 1.23E4
	double	1.23e4L
long	**double**	

C99+ **boolean** true false

May use **typedef** to define a type

```
typedef float* fptr
typedef long long ll
```

Operators

Arithmetic operators

Binary operators: + - * / % Mod/remainer operator (for signed/unsigned integers only)

Unary operators: ++ -- Increase/decrease by 1, after/before the current operation

Relational operators

> < >= <= == != Results: 1 (true) or 0 (false)

Logical operators

&& (and) || (or) ! (not)

Assignment operators

= += -= *= /= %= Result is the value of the left operand after assignment

Other operators

sizeof() & (address) * (dereference) , . ->

Bitwise operators

- **&** (bitwise AND)
- **|** (bitwise OR)
- **~** (bitwise NOT)
- **^** (bitwise XOR): 1 when the two bits are different
- **<<** (left shift)

```
0b01101011<<2     0b10101100
```

- **>>** (right shift)

```
0b01101011>>3     0b00001101
```

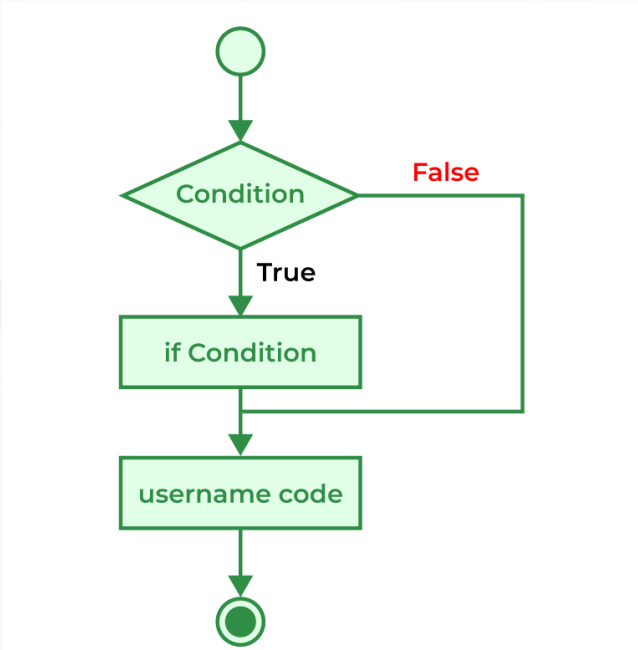
Operator Precedence

Precedence	Operator	Description
1	++ -- () [] . ->	Suffix/postfix increment and decrement Function call Array subscripting Structure and union member access Structure and union member access through pointer
2	! * & sizeof	Logical NOT Indirection (dereference) Address of … Size of …
3	* / %	Multiplication, division, and remainder
4	+ -	Addition and subtraction
5	< <= > >=	Relational operators < and ≤ respectively Relational operators > and ≥ respectively
6	== !=	Relational = and ≠ respectively
7	&&	Logical AND
8	||	Logical OR
9	= += -=	Simple assignment Assignment by sum and difference

Conditional statements

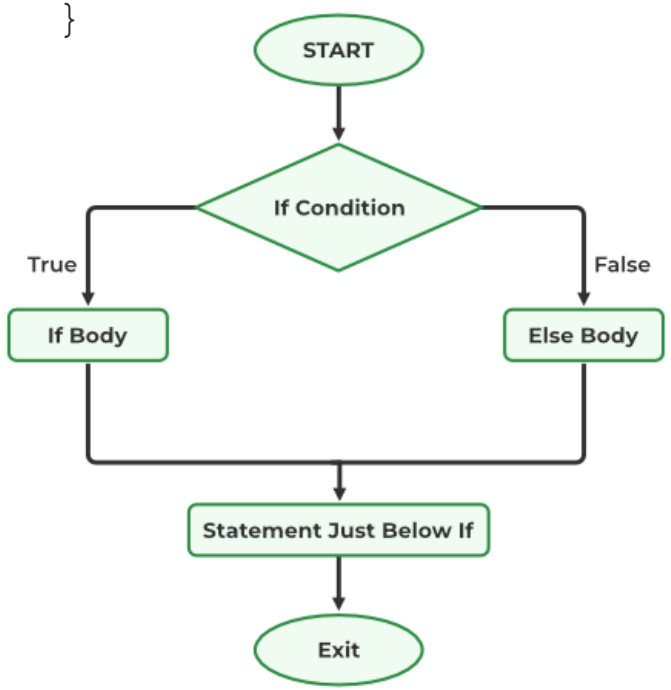
if statement

```
if(condition)
{
    statements;
}
```



if else statement

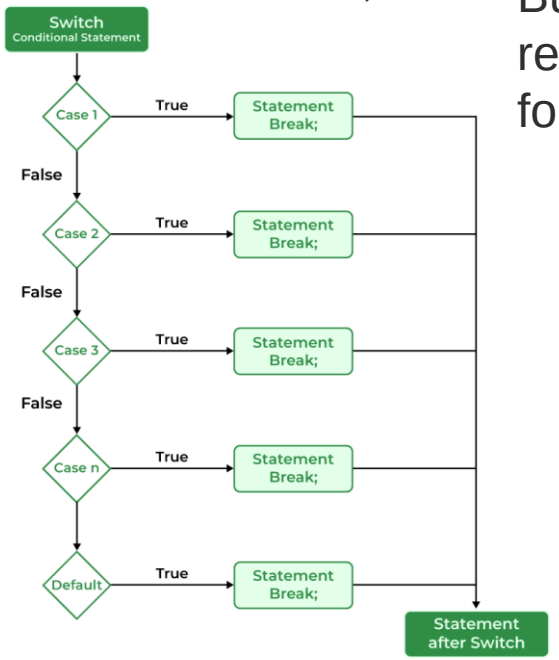
```
if(condition)
{
    statements;
}
else
{
    statements;
}
```



switch statement

```
switch(expression)
{
case 1:
    statements; break;
case 2:
    statements; break;
…
default:
    statements;
}
```

Optional.
But recommended for clarity

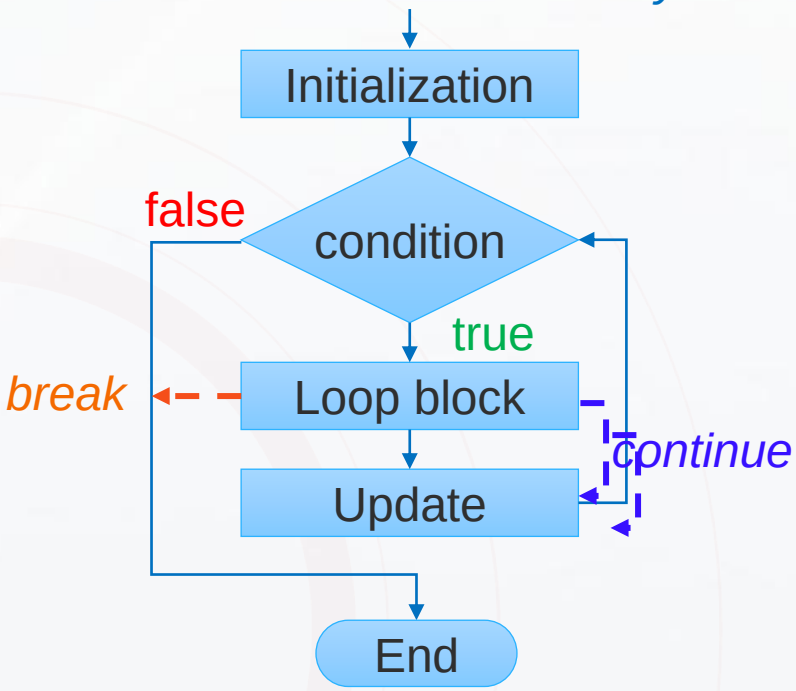


Loops

For loop

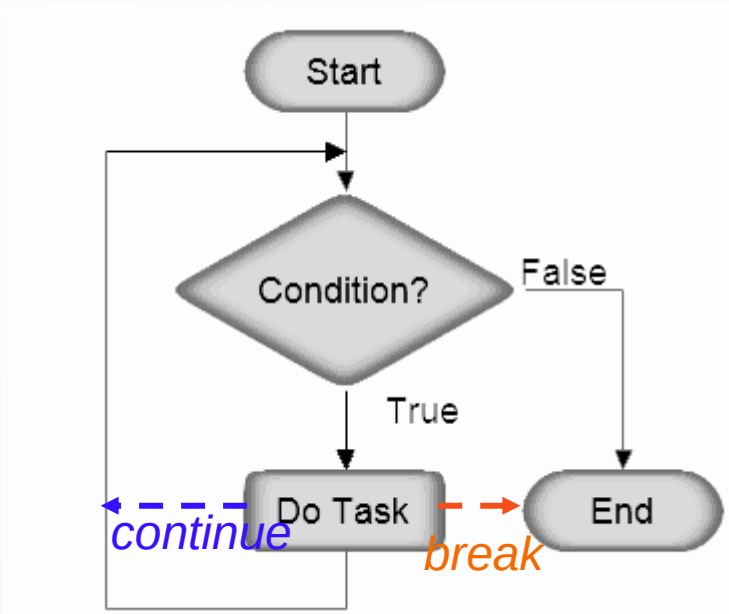
```
for(init; cond; update)
{
    statements;
}
```

May or may not run



While loop

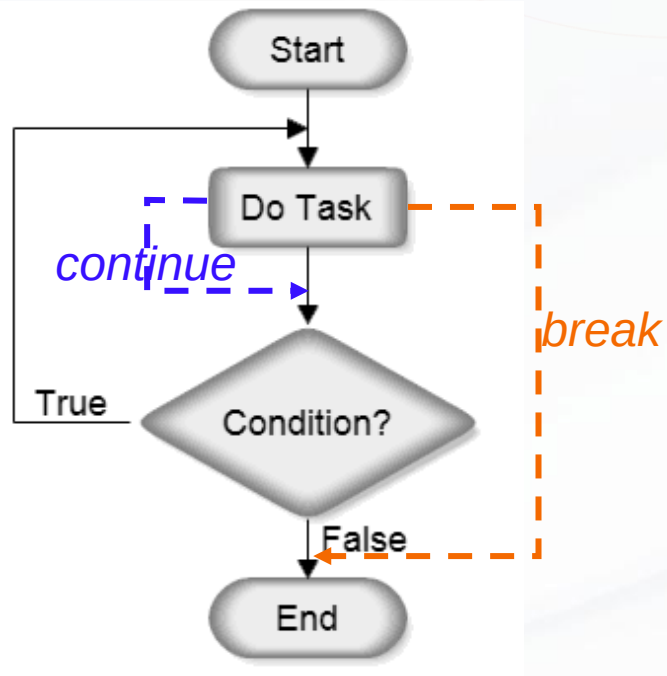
```
while(condition)
{
    statements;
}
```



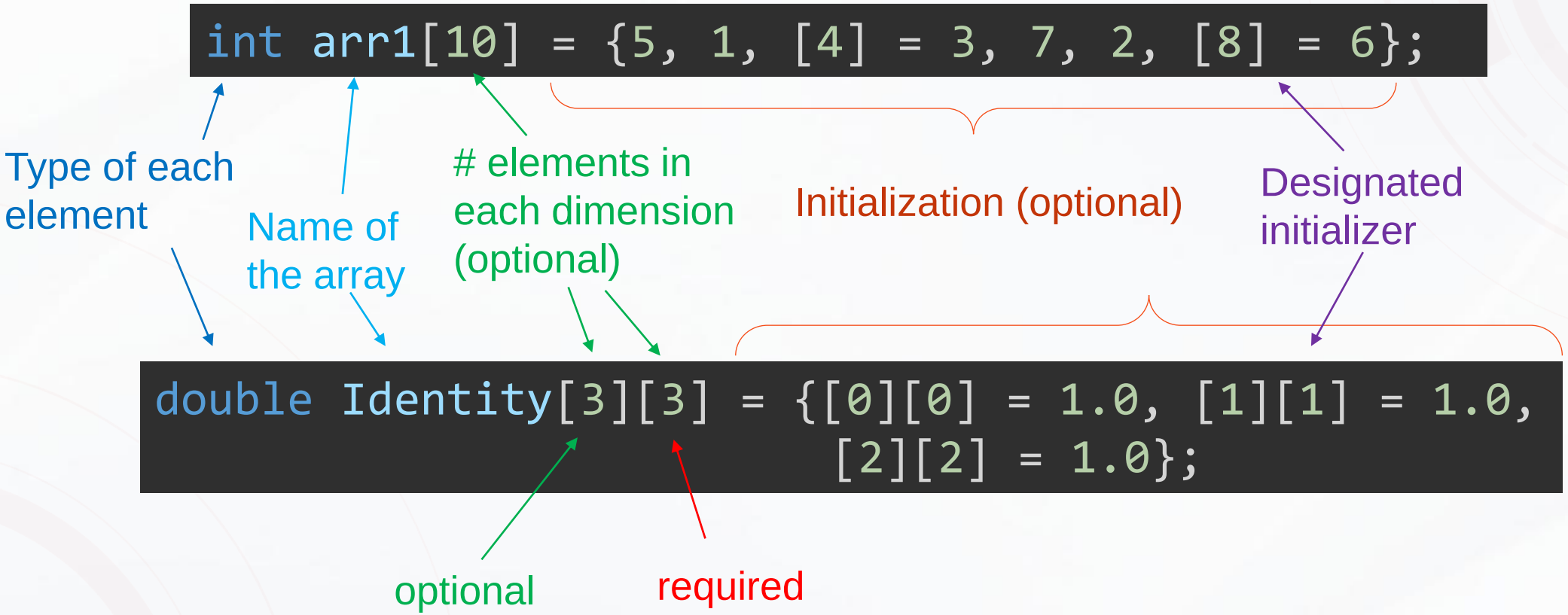
Do-while loop

```
do
{
    statements;
} while(condition);
```

Run at least once



Arrays



No arithmetic operation / assignment allowed on an entire array

	Pointer	Array
Declaration/ initialization	`int *ptr_to_x = &x;`	`int Arr1[] = {4, 2, 3};`

An array is like a pointer

- Accessing/writing content

`y = *ptr_to_x;`	`y = Arr1[0];`
`*ptr_to_x = 99;`	`Arr1[1] = 5;`
`ptr_to_x[0] = 99;`	`*(Arr1 + 1) = 5;`

- Passing address to a function

`scanf("%i", ptr_to_x);`	`scanf("%i", Arr1 + 1);`

An array is slightly different from a pointer

```
char *t = "How big is it?";
```

This creates an 8-byte pointer variable to store the address of the static string data

```
sizeof(t)
```
➡ 8

```
&t
```
➡ pointer of pointer

```
t[4] = 'B';
```
✗

Static data cannot be changed!

```
t = s;
```
✓

A pointer can be re-assigned

```
char s[] = "How big is it?";
```

This creates a 15-byte char array to store a copy of the string data

```
sizeof(s)
```
➡ 15

```
&s
```
➡ Pointer to the first elem. Same as s

```
s[4] = 'B';
```
✓

Content of an array can be changed

```
s = t;
```
✗

Name of an array cannot be changed!

Pointer decay: an array loses it length info when passed to a function.

struct

- Defining a struct (interface)

```
struct fish
{
    const char *name;
    const char *species;
    int teeth;
    int age;
};
```

- Contains data of different types
- Has fixed length
- Elements have distinct names

- Declaring/initializing an instance (variable)

```
    struct fish snappy = {"Snappy", "Piranha", 69, 4};
```

```
    struct fish carp = {.teeth=56};
```

- Assignments

```
struct fish snappy = {"Snappy", "Piranha", 69, 4};
gnasher = snappy;
```

struct

- Accessing the fields of a struct

```
    printf("Name = %s\n", snappy.name);
```

- struct pointers

```
struct turtle *t;
(*t).age++;
```

```
t->age ++;
```

- struct member may be an array (union, enum, struct …)

```
typedef struct {
    int number;
    char name[NAME_LEN+1];
    int on_hand;
} Part;
```

- struct/union arguments are copied when passed to a function
- Return value may be a struct/union

```
Part build_part(int number, …)
{
    Part p;…
    return p;
}
```

union

- All union members share the same memory. Only one member can be used!

```
typedef union {
    short count;
    float weight;
    float volume;
} quantity;
```

quantity (short or float)

enum

- An enum variable can take only the given values

```
typedef enum {MON=1, TUE, WED, THU, FRI, SAT, SUN} DAY;
```

Best for switch case structures!

Standard libraries - <string.h>

```
char *strcat(char *dest, const char *src)
char *strncat(char *dest, const char *src, size_t n)
char *strchr(const char *str, int c)
int strcmp(const char *str1, const char *str2)
int strncmp(const char *str1, const char *str2, size_t n)
char *strcpy(char *dest, const char *src)
char *strncpy(char *dest, const char *src, size_t n)
size_t strlen(const char *str)
char *strstr(const char *haystack, const char *needle)
```

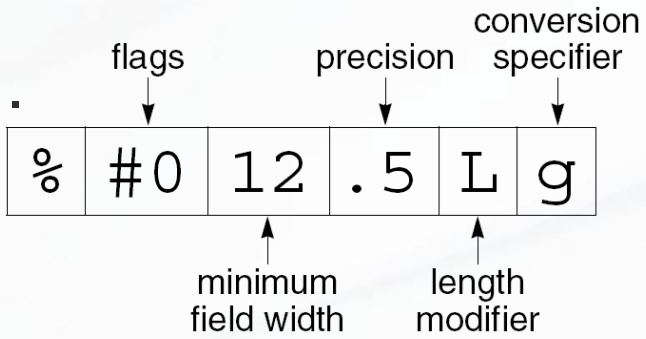
Standard libraries - <stdio.h>

```
int fprintf(FILE *stream,
            const char *format, ...);
int fscanf(FILE *stream,
            const char *format, ...);
```

```
printf(…)   ⟺   fprintf(stdout, …)
scanf(…)        fscanf(stdin, …)
```

Conversion specifiers in the format string

%d %i %o %u %x %f %e %E %g %%...



Standard libraries – <stdio.h> File I/O

- `fopen` returns a FILE pointer: (NULL pointer if fails)

```
FILE *fp;
fp = fopen("in.dat", "r");
```

- `fclose` closes a file that is no longer in use:

```
int fclose(FILE *fp);
```

String	Meaning	binary
"r"	For reading only	"rb"
"w"	For writing only (file may not exist)	"wb"
"a"	For appending only (file should exist)	"ab"
"r+"	For reading & writing (starting at beginning)	"rb+"
"w+"	For reading & writing (overwritten if file exists)	"wb+"
"a+"	For reading & writing (append if file exists)	"ab+"

File I/O functions in <stdio.h>

```
size_t fwrite(const void *ptr, size_t size,
              size_t nmemb, FILE *stream)
size_t fread(void *ptr, size_t size, size_t nmemb,
             FILE *stream)
int fseek(FILE *stream, long int offset, int whence)
```

SEEK_SET	Beginning of file
SEEK_CUR	Current file position
SEEK_END	End of file

```
long int ftell(FILE *stream)
void rewind(FILE *stream)
```

Standard libraries – <math.h>

```
double ceil(double x)
double floor(double x)
double fabs(double x)
double log(double x)
double log10(double x)
double fmod(double x, double y)
double sqrt(double x)
double pow(double x, double y)
double modf(double x, double *integer)
double exp(double x)
double cos(double x), sin, tan
double acos(double x), asin, atan
double atan2(double y, double x)
double sinh(double x), cosh, tanh
double asinh(double x), acosh, atanh
```

Thank you, and Happy Coding!