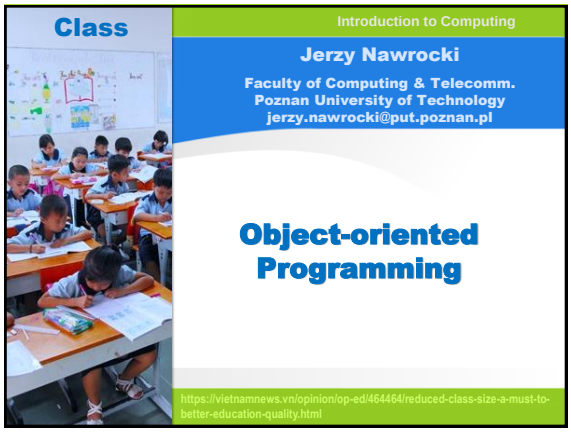
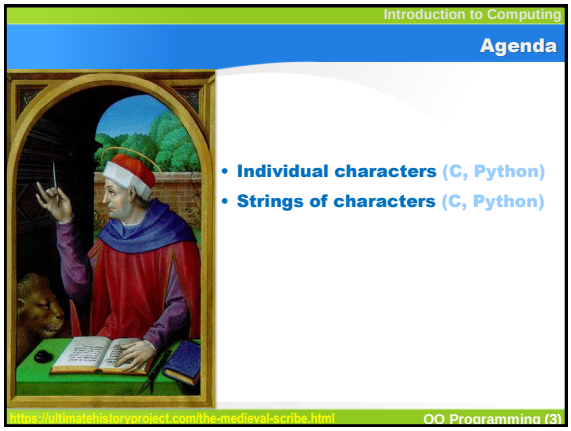
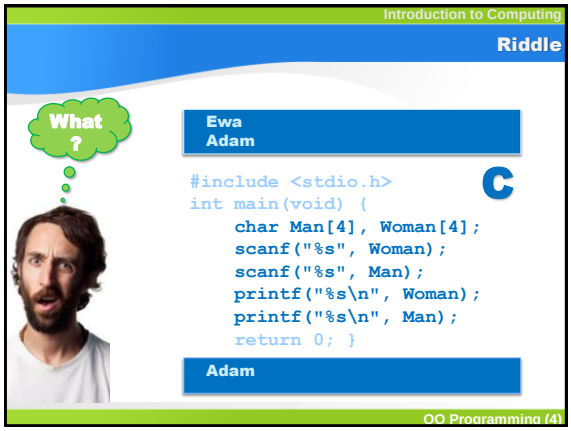
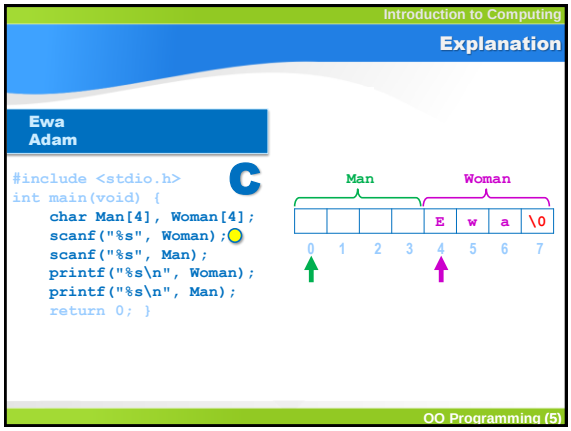
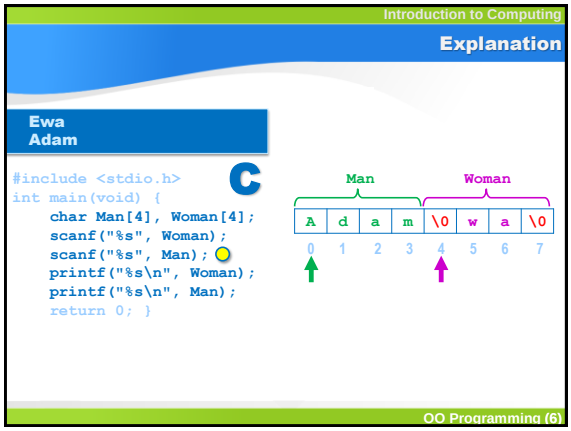

Class
Introduction to Computing
Jerzy Nawrocki
Faculty of Computing & Telecomm.
Poznan University of Technology
jerzy.nawrocki@put.poznan.pl
Object-oriented Programming
https://vietnamnews.vn/opinion/op-ed/464464/reduced-class-size-a-must-to-better-education-quality.html


Introduction to Computing
Jerzy Nawrocki
Faculty of Computing & Telecom.
Poznan University of Technology
jerzy.nawrocki@put.poznan.pl
Text processing
(continued)
https://ultimatehistoryproject.com/the-medieval-scribe.html


Introduction to Computing
Agenda
• Individual characters (C, Python)
• Strings of characters (C, Python)
https://ultimatehistoryproject.com/the-medieval-scribe.html
OO Programming (3)


Introduction to Computing
Riddle
What ?
Ewa
Adam
C
#include <stdio.h>
int main(void) {
char Man[4], Woman[4];
scanf("%s", Woman);
scanf("%s", Man);
printf("%s\n", Woman);
printf("%s\n", Man);
return 0; }
Adam
OO Programming (4)

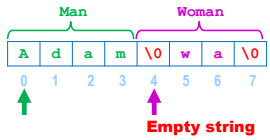

Introduction to Computing
Explanation
Ewa
Adam
C
#include <stdio.h>
int main(void) {
char Man[4], Woman[4];
scanf("%s", Woman);
scanf("%s", Man);
printf("%s\n", Woman);
printf("%s\n", Man);
return 0; }
Man
Woman
E w a \0
0 1 2 3 4 5 6 7
OO Programming (5)


Introduction to Computing
Explanation
Ewa
Adam
C
#include <stdio.h>
int main(void) {
char Man[4], Woman[4];
scanf("%s", Woman);
scanf("%s", Man);
printf("%s\n", Woman);
printf("%s\n", Man);
return 0; }
Man
Woman
A d a m \0 w a \0
0 1 2 3 4 5 6 7
OO Programming (6)

Explanation

Ewa
Adam

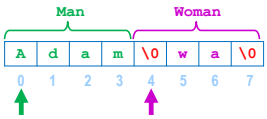
```
#include <stdio.h>
int main(void) {
    char Man[4], Woman[4];
    scanf("%s", Woman);
    scanf("%s", Man);
    printf("%s\n", Woman);
    printf("%s\n", Man);
    return 0; }
```



Explanation

Ewa
Adam

```
#include <stdio.h>
int main(void) {
    char Man[4], Woman[4];
    scanf("%s", Woman);
    scanf("%s", Man);
    printf("%s\n", Woman);
    printf("%s\n", Man);
    return 0; }
```



Adam

Agenda

- Individual characters (C, Python)
- Strings of characters (C, Python)
- **Rule-based text processing** (AWK)

https://ultimatehistoryproject.com/the-medieval-scribe.html

Rule-based programming

```
Jerzy Nawrocki    43089  I1
Jane  Kowalski    43780  I2
Adam  Malinowski  43990  I1
```

```
BEGIN      { print "-----"; }
$4=="I2"   { print $2, $1; }
END        { print "*****"; }
```

```
-----
Kowalski Jane
*****
```


Rule-based programming

```
Jerzy Nawrocki    43089  I1
Jan   Kowalski    43780  I2
Adam  Malinowski  43990  I1
```

```
END        { print "*****"; }
$4=="I2"   { print $2, $1; }
BEGIN      { print "-----"; }
```

```
-----
Kowalski Jan
*****
```


Agenda

- Individual characters (C, Python)
- Strings of characters (C, Python)
- Rule-based text processing (AWK)
- **Regular expressions** (AWK, Python, C)

https://ultimatehistoryproject.com/the-medieval-scribe.html

Regular expressions

Arithmetic expressions

Value: Text → Number

Value(2·3 + 3) = 9

Regular expressions

Value: Text → SetOfCharacterStrings

Value(/Ala | Ola/) = {"Ala", "Ola"}

Patterns with regular expressions

It's a favourite project of mine,
A new value of π to assign.
I would fix it at 3,
For it's simpler, you see,
Than 3 point 1 4 1 5 9

Substring *String ~ / reg_exp /*

$0 ~ /ne/

It's a favourite project of mine,
A new value of π to assign.

Patterns with regular expressions

It's a favourite project of mine,
A new value of π to assign.
I would fix it at 3,
For it's simpler, you see,
Than 3 point 1 4 1 5 9

Substring *String ~ / reg_exp /*

/ne/

It's a favourite project of mine,
A new value of π to assign.

$0 ~ / wyr_reg / = / wyr_reg /

Quiz

What will happen?

```
1
22
.
A
H2O
```

/^.$/

```
1
.
A
```


Riddle

Does 22*3 equal 2+3 ?

Example

What will happen?

```
1
22
.
A
H2O
```

/[0123456789]/

```
1
22
H2O
```

Example

What will happen?

```
1
22
.
A
H2O
```

```
/[0-9]/
```

```
1
22
H2O
```


Complement of a set of characters

[^ ...]

Example

What will happen?

```
1
22
.
A
H2O
```

```
/[^0-9]/
```

```
.
A
H2O
```


Example

What will happen?

```
1
22
.
A
H2O
```

```
/^[0-9]/
```

```
1
22
```


The `next` instruction

What will happen?

```
1
22
.
A
H2O
```

```
/^[0-9]/  {print "d..";}
/^.$/     {print $1;}
```

```
d..
1
d..
.
A
```


The `next` instruction

What will happen?

```
1
22
.
A
H2O
```

```
/^[0-9]/  {print "d..";
           next; }
/^.$/     {print $1;}
```

```
d..
d..
.
A
```

Disjunction (or)

reg_exp | *reg_exp*

What this program is doing?

It's a favourite project of mine,
A new value of π to assign.
I would fix it at 3,
For it's simpler, you see,
Than 3 point 1 4 1 5 9

/im | in/

Disjunction (or)

reg_exp | *reg_exp*

What this program is doing?

It's a favourite project of mine,
A new value of π to assign.
I would fix it at 3,
For it's simpler, you see,
Than 3 point 1 4 1 5 9

/im | in/

It's a favourite project of mine,
For it's simpler, you see,
Than 3 point 1 4 1 5 9

Parentheses and disjunction

Parentheses change precedence of operators:
3*(4 + 5) = 3* 9 = 27

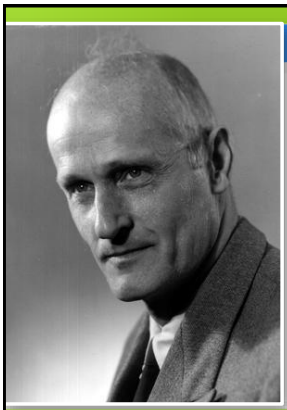
Distributive law:
3*(4 + 5) = 3*4 + 3*5 = 12 + 15 = 27
(4 + 5)*3 = 4*3 + 5*3 = 12 +15 = 27

Distributivity of concatenation over disjunction:

/i(m | n)/ = /im | in/

/Lond(o | y)n/ = /London | Londyn/

Stephen Kleene



r+ = r | rr | rrr| …

1+ = { 1, 11, 111, 1111, … }

http://www.math.wisc.edu/~gpslogic/

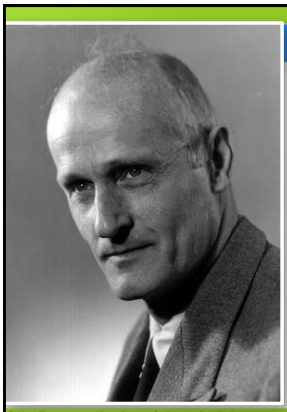
Example

```
Jurek 10  Adam  200
Params: 3  40  -5
```

```
    {for (i=1; i<=NF; i++)
        if ($i ~ /^[0-9]+$/) num++;}
END {print "Found "num" numbers.";}
```

```
Found 4 numbers.
```


Kleene's star



r+ = r | rr | rrr| …

ε = empty string

x ε = x

ε x = x

r* = ε | r | rr | rrr| …

http://www.math.wisc.edu/~gpslogic/

Kleene's star

r (ε | r | rr | rr |..) =
r | rr | rrr | rrrr |..

r+ = r r* = r* r

x w* = x | x w+

Example

```
    {for (i=1; i<=NF; i++)
        if ($i ~ /^[0-9]+$/) num++;}
END {print "Found "num" numbers.";}
```

```
    {for (i=1; i<=NF; i++)
        if ($i ~ /^[0-9][0-9]*$/) num++;}
END {print "Found "num" numbers.";}
```


Riddle

Does 22*3 equal 2+3 ?

Riddle

22* = 2+

Does 22*3 equal 2+3 ?

At least one '2'

2 ... 2 3

Class

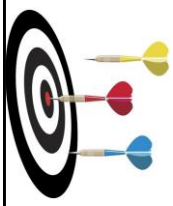
Jerzy Nawrocki

Faculty of Computing & Telecomm.
Poznan University of Technology
jerzy.nawrocki@put.poznan.pl

Object-Oriented Programming

https://vietnamnews.vn/opinion/op-ed/464464/reduced-class-size-a-must-to-better-education-quality.html

Aim of the lecture



Help the students to understand the paradigm of object-oriented programming.

Agenda

- Records
- Classes
- Dynamic storage allocation
- Lists

Records vs. arrays

Person

Name	"Einstein"
Born	1879

Name	Born
"Einstein"	1879

Accessing record fields in C

Person

Born	
Cash	

Einstein

Born	1879
Cash	

RecordVar . Field

C

```
#include <stdio.h>
struct Person {int Born;
               int Cash;
              };
int main(void){
    struct Person Einstein;
    Einstein.Born= 1879;
    printf("%d", Einstein.Born); }
```


Accessing dictionary fields in Python

Person

Born	
Cash	

Einstein

Born	1879
Cash	

DictionaryVar [Field]

```
Einstein= {"Born": 0,
           "Cash": 0}
Einstein["Born"]= 1879
print(Einstein["Born"])
```


Dictionary

{ key : value ,
key : value ,
. . .
key : value }

Accessing dictionary fields in Python

```
Einstein= {"Born": 0,
           "Cash": 0}
Einstein["Born"]= 1879
print(Einstein["Born"])
```

≡

```
Einstein= {1: 0,
           2: 0}
Einstein[1]= 1879
print(Einstein[1])
```

A puzzle: What will the result be?

```
data= [{"Id": 0}]*10

data[0]["Id"]= 1


data[1]["Id"]= 2
```


A puzzle: What will the result be?

```
data= [{"Id": 0}]*10

data[0]["Id"]= 1
print(data[0]["Id"])

data[1]["Id"]= 2
print(data[0]["Id"])
print(data[1]["Id"])
```


Surprise

```
1.  data= [{"Id": 0}]*10
2.  data[0]["Id"]= 1
3.  print(data[0]["Id"])
4.
5.  data[1]["Id"]= 2
6.  print(data[0]["Id"])
7.  print(data[1]["Id"])
```

stdout

```
1
2
2
```


Python's lists (~ arrays) are dynamic

```
List= []
. . .
List.append(new item)
```


Students with the highest grade

Id

Grade

```
111  5
112 10
113  8
114 10
```

?

```
112 10
114 10
```


Students with the highest grade

```
111  5
112 10
113  8
114 10
```

```
import sys


for line in sys.stdin:
    i, g = line.split()
    i= int(i)
    g= int(g)
```

Students with the highest grade

```
111  5
112 10
113  8
114 10
```

```
import sys

max= 0
for line in sys.stdin:
    i, g = line.split()
    i= int(i)
    g= int(g)

    if g > max:
       max= g
```


Python's lists (~ arrays) are dynamic

```
List= []
. . .
List.append(new item)
```


Students with the highest grade

```
111  5
112 10
113  8
114 10
```

```
import sys
data= []
max= 0
for line in sys.stdin:
    i, g = line.split()
    i= int(i)
    g= int(g)
    data.append({"Id": i, "Grade": g})
    if g > max:
       max= g
```

0	Id	111	Grade	5
1	Id	112	Grade	10
	...	...	...	...

Students with the highest grade

```
import sys
data= []
max= 0
for line in sys.stdin:
    i, g = line.split()
    i= int(i)
    g= int(g)
    data.append({"Id": i, "Grade": g})
    if g > max:
       max= g
for item in data:
    if item["Grade"] == max:
       print(item["Id"], item["Grade"])
```

0	Id	111	Grade	5
1	Id	112	Grade	10

C

```
111  5
112 10
113  8
114 10
```

```
#include <stdio.h>
int main(void){

    while (scanf("%d %d", &i, &g) != EOF){

        }

    }
```

C

```
111  5
112 10
113  8
114 10
```

```
#include <stdio.h>
int main(void){

    int max, k, i, g   ;
    max= 0;
    k= 0;
    while (scanf("%d %d", &i, &g) != EOF){

        if (g > max){
           max= g; }
        k+= 1;
        }

    }
```

```c
#include <stdio.h>
int main(void){
    struct item {int Id;
                 int Grade;};
    struct item data[100];
    int max, k, i, g   ;
    max= 0;
    k= 0;
    while (scanf("%d %d", &i, &g) != EOF){

       if (g > max){
          max= g; }
       k+= 1;
       }


    }
```

C

111 5
112 10
113 8
114 10

	Id	Grade
0		
1		
2		
3		
	...	...

```c
#include <stdio.h>
int main(void){
    struct item {int Id;
                 int Grade;};
    struct item data[100];
    int max, k, i, g   ;
    max= 0;
    k= 0;
    while (scanf("%d %d", &i, &g) != EOF){
       data[k].Id= i;
       data[k].Grade= g;
       if (g > max){
          max= g; }
       k+= 1;
       }


    }
```

C

111 5
112 10
113 8
114 10

	Id	Grade
0	111	5
1	112	10
2	113	8
3	114	10
	...	...

```c
#include <stdio.h>
int main(void){
    struct item {int Id;
                 int Grade;};
    struct item data[100];
    int max, k, i, g, j;
    max= 0;
    k= 0;
    while (scanf("%d %d", &i, &g) != EOF){
       data[k].Id= i;
       data[k].Grade= g;
       if (g > max){
          max= g; }
       k+= 1;
       }
    for (j= 0; j < k; j+= 1){
        if (data[j].Grade == max)
           printf("%d %d\n", data[j].Id, data[j].Grade);
        }
    }
```

C

	Id	Grade
0	111	5
1	112	10
2	113	8
3	114	10
	...	...

Introduction to Computing

List (Python) vs. arrays (C)

0	Id	111	Grade	5
1	Id	112	Grade	10
	...	...	...	...

dynamic

	Id	Grade
0	111	5
1	112	10
2	113	8
3	114	10
	...	...
99		

static

C

OO Programming (58)

Introduction to Computing

typedef

C

```c
typedef oldType newName;
```

```c
typedef int Natural;
Natural x, y;
```

OO Programming (59)

Introduction to Computing

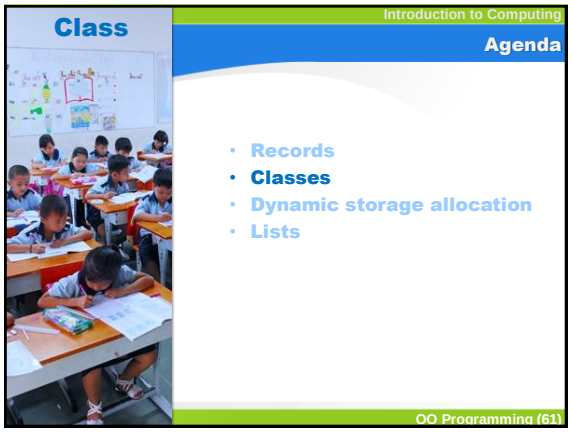
typedef

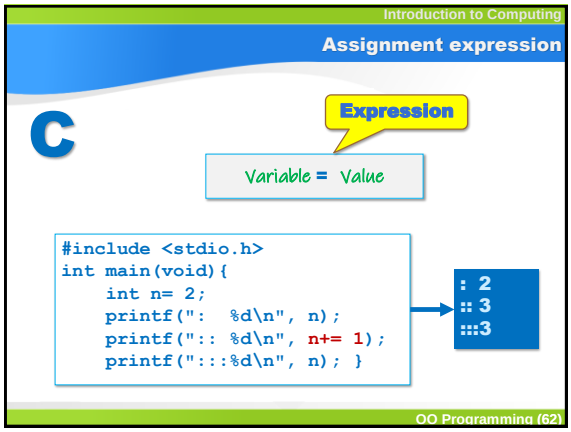
```c
typedef oldType newName;
```

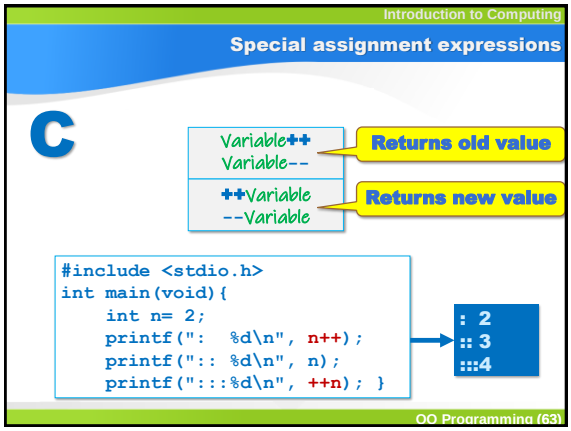
```c
struct Person {int Born;
               int Cash;
              };

struct Person Einstein;
Einstein.Born= 1879;
```

```c
struct Person {int Born;
               int Cash;
              };
typedef struct Person Men
Men Einstein;
Einstein.Born= 1879;
```

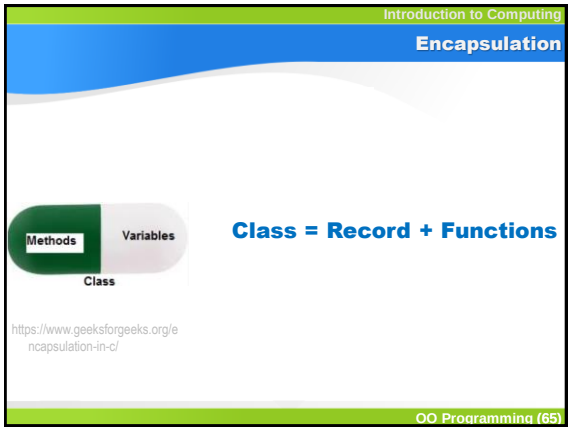
OO Programming (60)

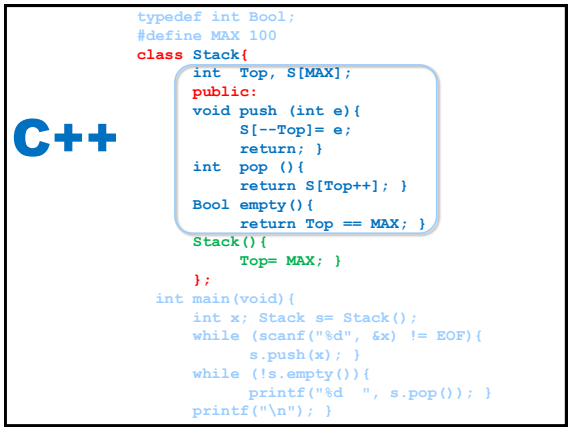

Class
Introduction to Computing
Agenda
Records
Classes
Dynamic storage allocation
Lists
OO Programming (61)


Introduction to Computing
Assignment expression
C
Expression
Variable = Value
#include <stdio.h>
int main(void){
int n= 2;
printf(": %d\n", n);
printf(":: %d\n", n+= 1);
printf(":::%d\n", n); }
: 2
:: 3
:::3
OO Programming (62)


Introduction to Computing
Special assignment expressions
C
Variable++
Variable--
Returns old value
++Variable
--Variable
Returns new value
#include <stdio.h>
int main(void){
int n= 2;
printf(": %d\n", n++);
printf(":: %d\n", n);
printf(":::%d\n", ++n); }
: 2
:: 3
:::4
OO Programming (63)


Introduction to Computing
Origins: Simula 67 (Norwegian Computing Center)
Kristen Nygaard
Ole-Johan Dahl
OO Programming (64)


Introduction to Computing
Encapsulation
Methods
Variables
Class
Class = Record + Functions
https://www.geeksforgeeks.org/encapsulation-in-c/
OO Programming (65)


C++
typedef int Bool;
#define MAX 100
class Stack{
int Top, S[MAX];
public:
void push (int e){
S[--Top]= e;
return; }
int pop (){
return S[Top++]; }
Bool empty(){
return Top == MAX; }
Stack(){
Top= MAX; }
};
int main(void){
int x; Stack s= Stack();
while (scanf("%d", &x) != EOF){
s.push(x); }
while (!s.empty()){
printf("%d ", s.pop()); }
printf("\n"); }

C++

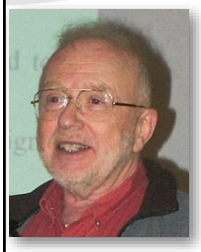
```
typedef int Bool;
#define MAX 100
class Stack{
      int  Top, S[MAX];
      public:
      void push (int e){
            S[--Top]= e;
            return; }
      int  pop (){
            return S[Top++]; }
      Bool empty(){
            return Top == MAX; }
      Stack(){
            Top= MAX; }
      };
  int main(void){
      int x; Stack s= Stack();
      while (scanf("%d", &x) != EOF){
            s.push(x); }
      while (!s.empty()){
            printf("%d  ", s.pop()); }
      printf("\n"); }
```


C++

```
typedef int Bool;
#define MAX 100
class Stack{
      int  Top, S[MAX];
      public:
      void push (int e){
            S[--Top]= e;
            return; }
      int  pop (){
            return S[Top++]; }
      Bool empty(){
            return Top == MAX; }
      Stack(){
            Top= MAX; }
      };
int main(void){
    int x; Stack s= Stack();
    while(scanf("%d",&x) != EOF)
          s.push(x);
    while(!s.empty())
          printf("%d  ",s.pop());
    printf("\n"); }
```

```
import sys
MAX= 100
class Stack:
    def __init__(self):
        self.S= [0]*MAX
        self.Top= MAX
    def Push(self, e):
        self.Top-= 1
        self.S[self.Top]= e
    def Pop(self):
        e= self.S[self.Top]
        self.Top+= 1
        return e
    def Empty(self):
        return self.Top == MAX

s= Stack()
for line in sys.stdin:
    for x in line.split():
        s.Push(x)
while not s.Empty():
    print(s.Pop(),"  ",end="")
```


Information hiding

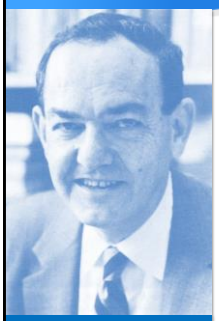


David Parnas
1941 -

Information hiding is characterized by the idea of **"secrets"**, design and **implementation decisions** that a software developer hides in one place from the rest of a program.

-- Steve McConnell
(IEEE Software, March 1996)

Cost of information



Herbert Simon

Recipient's **cost of information**
=
Recipient's **attention**

Number of decimal digits

```
int Digi (int x) {
   int r, p;
   r = 1;   p = 10;
   while (x >= p) {
          r = r + 1;
          p = p * 10;  }
   return r;
   }
```

Iterative version

```
int Digi (int n) {
   if (n < 10) return 1;
   else return 1 + Digi (n/10);
   }
```

Recursive version


```
import sys

class Stack:
    def __init__(self):
        self.S= []

    def Push(self, e):
        self.S.append(e)

    def Pop(self):
        e= self.S[len(self.S)-1]
        self.S.pop(len(self.S)-1)
        return e
    def Empty(self):
        return len(self.S) == 0

s= Stack()
for line in sys.stdin:
    for x in line.split():
        s.Push(x)
while not s.Empty():
    print(s.Pop(),"  ",end="")
```

```
import sys
MAX= 100
class Stack:
    def __init__(self):
        self.S= [0]*MAX
        self.Top= MAX
    def Push(self, e):
        self.Top-= 1
        self.S[self.Top]= e
    def Pop(self):
        e= self.S[self.Top]
        self.Top+= 1
        return e
    def Empty(self):
        return self.Top == MAX

s= Stack()
for line in sys.stdin:
    for x in line.split():
        s.Push(x)
while not s.Empty():
    print(s.Pop(),"  ",end="")
```

Inheritance

An inheriting class (son)
extends
the inherited class (father)

Access specifiers

public	*name*
private	*__name*
protected	*_name*

C++

Inheritance in C++ (1 of 2)

```
extern "C" {
#include <stdio.h>
}
typedef int Bool;
#define MAX 100
class Stack{
      protected:
      int  Top, S[MAX];
      public:
      void push (int e){
            S[--Top]= e;
            return; }
      int  pop (){
            return S[Top++]; }
      Bool empty(){
            return Top == MAX; }
      Stack(){
            Top= MAX; }
      };
```

Subclass of Stack

```
class WindowStack: public Stack{
      protected:
      int Window;
      public:
      void Start(){
            Window= Top;
            return; }
      int  LookUp(){
            return S[Window++];}
      Bool Bottom(){
            return Window == MAX;}
      WindowStack(){
            return;}
      };
```


Inheritance in C++ (2 of 2)

C++

```
int main(){
    int x;
    WindowStack s= WindowStack();
    while(scanf("%d",&x) != EOF)
        s.push(x);
    s.Start();
    printf("%d  ", s.LookUp());
    printf("%d\n", s.LookUp());
    while(!s.empty())
        printf("%d  ",s.pop());
    printf("\n"); }
```


Inheritance in Python (1 of 2)

```
import sys
MAX= 100
class Stack:
   def __init__(self):
       self.S= [0]*MAX
       self.Top= MAX
   def Push(self, e):
       self.Top-= 1
       self.S[self.Top]= e
   def Pop(self):
       e= self.S[self.Top]
       self.Top += 1
       return e
   def Empty(self):
       return self.Top == MAX
```

Subclass of Stack

```
class WindowStack(Stack):
   def __init__(self):
       super().__init__()
       self.Window= 0
   def Start(self):
       self.Window= self.Top
   def LookUp(self):
       TMP= self.S[self.Window]
       self.Window += 1
       return TMP
   def Bottom(self):
       return self.Window == MAX
```


Inheritance in Python (2 of 2)

```
s= WindowStack()
for line in sys.stdin:
   for x in line.split():
      s.Push(x)
s.Start()
print(s.LookUp())
print(s.LookUp())
while not s.Empty():
     print(s.Pop(),"  ",end="")
```

Results

Success #stdin #stdout 0.02s 9084KB

stdin

1 2 3 4

stdout

4

3

4 3 2 1

LookUp()

LookUp()

Pop() Pop() ...

Class

Agenda

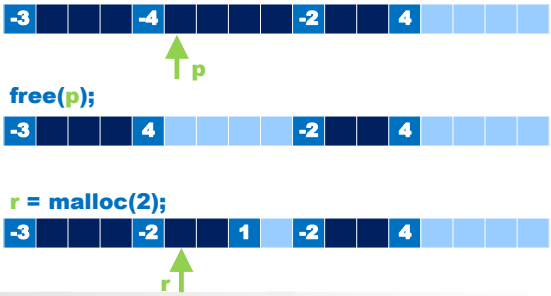
- Records
- Classes
- **Dynamic storage allocation**
- Lists

Dynamic storage allocation (DSA)

C

```
int *pointer;
pointer= malloc(size);
free(pointer);
```


First-fit strategy



Memory fragmentation

```
int *r;
r= malloc(5);
```

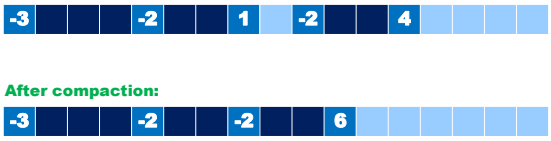
C

- There is **no** free block of size ≥ s.
- The **total size** of free blocks ≥ s.

Memory compaction

```
int *r;
r= malloc(5);
```

C



After compaction:

Compaction problem

Before compaction:

Compaction problem

After compaction:

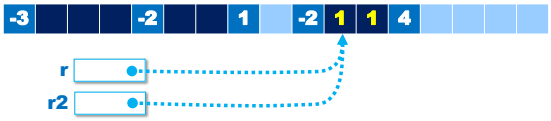
Reference variables must be updated.

Compaction problem

```
int *r, r2;
. . .
r2= r;
```

C

Before compaction:



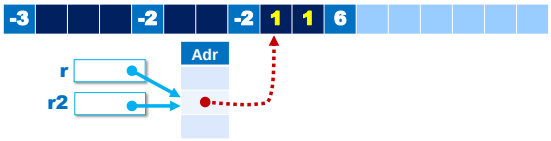
Problem: Many reference variables.

Indirect addresses

Before compaction:

Indirect addresses

After compaction:

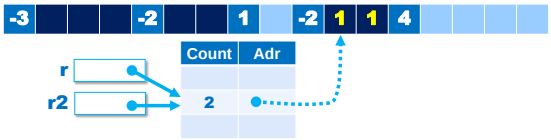


Only one address update per block.

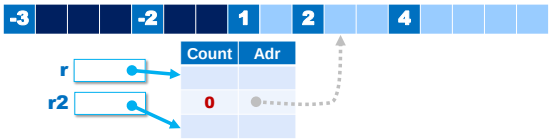
DSA with garbage collection

```
int *pointer;
pointer= malloc(size);
free(pointer);
```

Reference counting

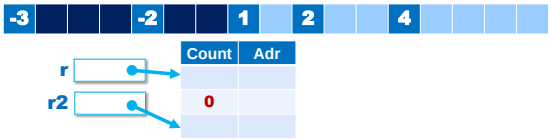


Reference counting



If Count == 0, free the block.

Reference counting



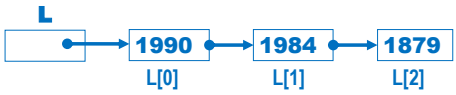
If Count == 0, free the block.

Agenda

- Records
- Classes
- Dynamic storage allocation
- **Lists**

Riddle

```
L= [1990, 1984, 1879]
print(L[0])
```



Lists in Python

[*item* , *item* , ... *item*]

```
L= [1990, 1984, 1879]
len(L)
L.insert(index, value)
L.pop(index)
L.remove(value)
L.sort()
L.reverse()
```

Lists in Python

[*item* , *item* , ... *item*]

```
L= [1990, 1984, 1879]
len(L)
L.insert(index, value)  ····▶ L.append(value)
L.pop(index)  ··········▶ L.pop()
L.remove(value)           del L
L.sort()
L.reverse()
```


Lists in Python

[*item* , *item* , ... *item*]

```
L= [1990, 1984, 1879]
len(L)
L.insert(index, value)  ····▶ L.append(value)
L.pop(index)  ··········▶ L.pop()
L.remove(value)           del L
L.sort()
L.reverse()

L2= L
L3= L.copy()
```

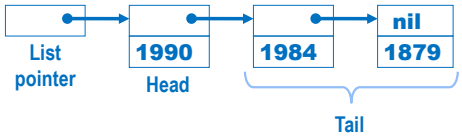

Pointers

C

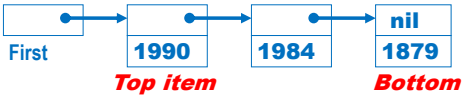
```
#include <stdio.h>
int main(void){
    int A, B, *Adr;
    A=3;
    B=4;
    Adr= &A;     // *Adr == A
    B= B + *Adr;
    printf("%d %d\n", A, B);}
```

```
3 7
```


List of 3 items (1990, 1984, 1879)



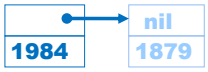
Stack



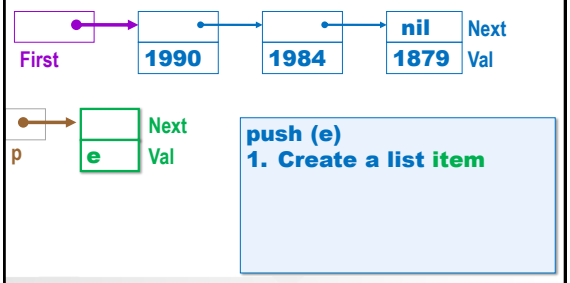
Stack implemented as a list

```
typedef struct Elem Item;
struct Elem {int Val;
             Item *Next;
             };
Item *First;
```

C

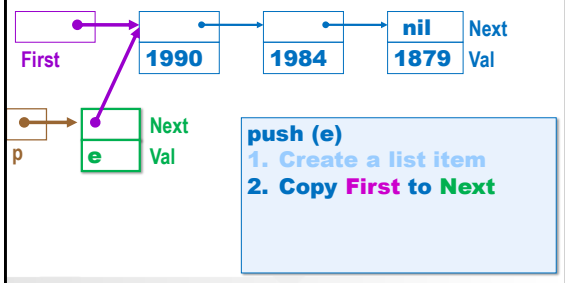


Stack as a list: push(e)



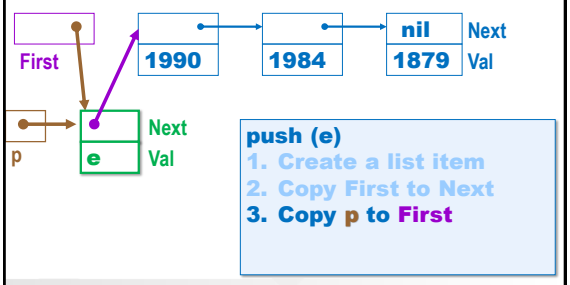
push (e)
1. Create a list item

Stack as a list: push(e)



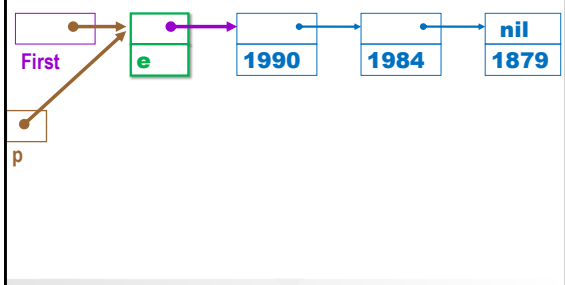
push (e)
1. Create a list item
2. Copy First to Next

Stack as a list: push(e)



push (e)
1. Create a list item
2. Copy First to Next
3. Copy p to First

Stack as a list: push(e)



Stack as a list: push(e)

push (e)
1. Create a list item

C

```
#include <stdlib.h>
typedef struct Elem Item;
struct Elem {int Val;
             Item *Next;
             };
Item *First;
#define ItemS sizeof(Item)
void push (int e){
     Item *p;
     p= malloc(ItemS);
     p->Val= e;
     (*p).Next= First;
     First= p;
     return;
     }
```

Number of bytes

Stack as a list: push(e)

push (e)
1. Create a list item
2. Copy First to Next

C

```
#include <stdlib.h>
typedef struct Elem Item;
struct Elem {int Val;
             Item *Next;
             };
Item *First;
#define ItemS sizeof(Item)
void push (int e){
     Item *p;
     p= malloc(ItemS);
     p->Val= e;
     (*p).Next= First;
     First= p;
     return;
     }
```

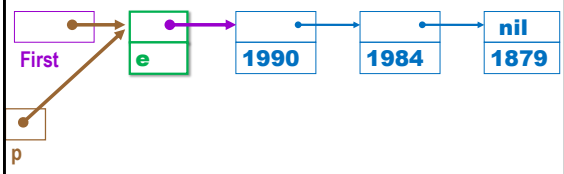
Stack as a list: push(e)

push (e)
1. Create a list item
2. Copy First to Next
3. Copy p to First

C

```
#include <stdlib.h>
typedef struct Elem Item;
struct Elem {int Val;
             Item *Next;
             };
Item *First;
#define ItemS sizeof(Item)
void push (int e){
     Item *p;
     p= malloc(ItemS);
     p->Val= e;
     (*p).Next= First;
     First= p;
     return;
     }
```


Stack as a list: push(e)



Stack as a list: push(e)

```
int pop (){
     Item *p; int e;
     p= First;
     First= (*p).Next;
     e= (*p).Val;
     free(p);
     return e;
     }
```

Assumption:
Stack is non-empty

C

```
#include <stdlib.h>
typedef struct Elem Item;
struct Elem {int Val;
             Item *Next;
             };
Item *First;
#define ItemS sizeof(Item)
void push (int e){
     Item *p;
     p= malloc(ItemS);
     p->Val= e;
     (*p).Next= First;
     First= p;
     return;
     }
```


Summary

Agenda

- Records
- Classes
- Dynamic storage allocation
- Lists