Тема 1. Променливи, указатели и референции, предаване на параметри

Организация:

ФОРМА НА СЕМЕСТРИАЛНИЯ КОНТРОЛ	Точки – К1
Контролни и/или тестове по време на лекции (2 теста на лекции)	до 30
Препитване (контролни) по раздели по време на лабораторни упражнения	до 40
Задача върху двата раздела (домашни задачи)	до 25
Активно участие по време на лабораторните упражнения	до 5
Общо	100

Форма на контрол	Точки – К2
Изпит - писмен със събеседване	100

Окончателна оценка в точки: К= 0,4 ×К1+ 0,6 ×К2

Организация:

- Литература (Основна)
- [1] Вл. Николов, Обектно-ориентирано програмиране – 1 част-Записки, ISBN 954-20-0324-2, Печатна база при Технически университет – Варна, 2005.
- [2] Вл. Николов, М. Карова, П. Владимирова, Н, Миндов Обектно-ориентирано програмиране – 1 част-Ръководство за лабораторни упражнения, ISBN 954-20-0325-0, Печатна база при Технически университет – Варна, 2005.
- [3] Вл. Николов, Обектно-ориентирано програмиране – 1 част - Сборник от тестове и задачи (Ръководство за курсов проект), Технически университет – Варна, 2013.

Съдържание на курса
Основни раздели

Първа част

Тема 1. Въведение в дисциплината.

Тема 2. Обекти и класове.

Тема 3. Производност и наследяване.

Тема 4. Полиморфизъм. Виртуални член функции.

Тема 5. Предефиниране на функции. Оператори. Шаблони.

Тема 6. Статични членове. Типови преобразувания. Изключения.

Тема 7. Именни пространства. Вградени обекти и функции. Подразбиращи се аргументи

Контролна работа

Съдържание на курса
Основни раздели

Втора част:

Тема 8. Стандартна темплейтна библиотека **STL**.

Тема 9. Последователни контейнери.

Тема 10. Итератори на контейнери.

Тема 11. Обобщени алгоритми.

Тема 12. Асоциативни контейнери

Тема 13. Адаптери. Видове.

Контролна работа

Тема 14. Съвместна работа на компонентите. Оценка на ефективността.

Съдържание на лекцията

- Характеристики на променливите;
- Статично свързвани променливи;
- Динамично свързвани променливи;
  - Стеково базирани;
  - Динамични;
- Указател;
- Референция;
- Предаване на параметри:
  - Предаване по стойност;
  - Предаване чрез референция. Константни параметри;
- Интелигентни указатели;

Променливи, характеристики на променливите

- **име (name);**
- **адрес (address)** *l - value* ;
- **Размер (size) ;**
- **Тип (type)** ;
- **Стойност (value)** ***r*** **-** *value*;
- **Време за съществуване (lifetime);**
- **Област на видимост (scope);**

Пример

Операторът за декларация на променлива:

int iVar = 57;

има следните характеристики:

Характеристики:

- Име: ***iVar;***
- Тип: ***int ;***
- Размер: ***sizeof(int)*** **-** Размерът на променливата се получава с помощта на функцията на езика ***sizeof;*** с параметър C++ тип.
- Стойност (начална): ***57;***
- Адрес на променливата: ***получава се по време на изпълнение;***

Характеристики:

- **Време за съществуване***: в зависимост от мястото на декларация (за глобални променливи – от стартиране до завършване на приложението);*
- **Област на видимост**: *в зависимост от мястото на декларация (за глобални променливи – от всички функции на приложението)* ;

Характеристики:

- **Тип** и **размер**, се свързват към нея по време на компилация (*static binding)*.
- **Адрес** и **стойност**, се свързват по време на изпълнение *динамично свързване* (*dynamic binding)*.

Статично свързвани променливи (*static binding*).

Дефиниции:

Статична памет ;

Статично свързвани променливи ;

Пример за статично свързана променлива:

```
1. int giVar=0;
2. int main() {
3.   giVar = 25;
4. // други оператори
5. }
```

Динамично свързвани променливи (*dynamic binding)*

Дефиниции:

- *стек (stack) ;*
- *динамична памет (heap);*

Динамично свързвани променливи (*dynamic binding)*

Два вида динамично свързвани променливи:

- **Динамично създавани променливи в стека** - стеково базирани (*stack based variables*);
- **Динамично създавани променливи в динамичната памет**-динамични (*heap based variables*);

Динамично свързвани променливи (*dynamic binding)*

Стеково базирани променливи (*stack based variables*);

Дефиниция: *Стекът* е област от паметта, която процесорът използва директно за записване на данни по време на изпълнение на програмата.

Динамично свързвани променливи (*dynamic binding)*

Примери за стеково базирани променливи:

```
1. void f() {
2. int iVar1;
3. // оператори
4. {
5.   int iVar2=10;
6. // оператори
7. }
8.}
```

Динамични променливи (*heap based variables*);

Особености на динамичните променливи:

- Паметта е на операционната с-ма (ОС);
- Представят се чрез указател, получен от ОС;
- Изцяло под програмния контрол на програмиста - команди;
  - Експлицитно създаване на паметта **new**;
  - Експлицитно освобождаване **delete**.

Динамични променливи (*heap based variables*);

Пример за динамична променлива:

```
1. void f() {
2. double *pdVar3 = new double;
3. *pdVar3 = 1.56;
4. // оператори (функции), използващи
   динамичната променлива
5. delete pdVar3;
6. }
```

Указател (pointer)

Дефиниция:

Указателят е проста променлива, която служи за съхраняване адреса на друга променлива от произволен тип.

Пример за деклариране на указател:

Операторът на C++:

```
int* piVar;
```

Указател (pointer)

При указателите има също:

лява стойност (*l*-value) и

стойност value (*r*-value).

Пример:

Дадена е C++ програма, която съдържа следните дефиниции на глобални променливи:

```
int i = 57;
int j = 31;
int* p = 0;
int* q = (int*) 1004;
```

Указател (pointer)

Указател (pointer)

Действие на оператора за присвояване:

i = j;

- Присвояване на стойности;
- След присвояването двете променливи имат еднакви стойности (31), равни на дясната страна.

p = q;

- Присвоява стойността на q (адрес), в примера 1004 - (записва стойността в паметта на адреса на p, в случая 1008);
- След присвояването двете променливи сочат една и съща памет (1004);

Указател (pointer)

Дереферииране на указатели

- За достъп до променливата, към която указателят сочи, е необходимо ***дереферииране*** на указател.
- Ако *q се използва в контекст, при който се очаква *r*-value, тогава се взема *r*-value на променливата, към която q сочи.
- Присвоявания

```
i = *q;
*q = 31;
```

Референция

Дефиниция:

В C++ *референцията* дефинира алтернативно име за достъп до декларирана променлива.

Синтаксис ***T&***

```
int iVar = 57;
int& refiVar = iVar;
```

Референция

В C++ всички референции трябва да се инициализират. В този случай, референцията refiVar *реферира променливата* iVar.

Предаване на параметри

Дефиниция:

Предаване на параметри определя методите, по които се предават данни между функците.

Два метода за предаване на параметри -- *предаване на стойност* -- *(pass-by-value)* и *предаване чрез референция* (*pass-by-reference*).

Предаване на параметри

- **Предаване по(на) стойност**

Дефиниции:

- Аргументите, специфицирани в обръщението към функцията се наричат ***актуални параметри*.**
- Аргументите, които се записват в дефиницията на функцията се наричат ***формални параметъри*.**
- ***Предаване по стойност (pass-by-value)*** е метод за предаване на параметри, при който ***актуалните параметри*** на функцията се копират във ***формалните*** *параметри* на функцията по време на обръщението

Предаване на параметри по стойност

Пример:

```
void F2 (int _ix)//дефиниция с формален параметър _ix
{
  _ix = 2;
  cout << _ix << endl;
}
void F1 ()
{
  int iy = 1;
  F2 (iy);     //обръщение с актуален параметър iy
  cout << iy << endl;
}
```

Предаване на параметри по стойност

Изход:

2

1

Предаване на параметри чрез референция

Дефиниция: Предаване чрез референция (pass-by-reference) е метод за предаване на параметри, при който *формалните параметри* на функцията са алтернативни имена на (по същество указатели към) съответните фактически параметри по време на обръщението към функцията.

Предаване на параметри чрез референция

Пример:

Програма 2 дефинира същите функции с използване на метода за предаване на параметри чрез референция.

```
void F2 (int & _ix) // референция _ix като формален параметър
{
  _ix = 2;
  cout << _ix << endl;
}
void F1 ()
{
  int iy = 1;
  F2 (iy); //обръщение с актуален параметър iy (“по адрес”)
  cout << iy << endl;
}
```

Предаване на параметри чрез референция

Изход от примерната програма:

2

2

Особености при предаване на параметрите

- ***Предаването по стойност*** създава локална променлива в стека и инициализира тази локална променлива с копие на стойността на фактическия параметър в стека.
- ***Предаването по референция*** не създава локални променливи, не инициализира (копира) такива за съхраняване на фактическия параметър. Поради начина, по който е имплементирана, *формалният параметър референция се използва за достъп до съответния фактически параметър*. Недостатъци? Предимства?

Предаване на параметри чрез референция - Константни параметри

```
void F2 (const int& _x)
{
    _x = 2; // Недопустимо, грешка при
       // компилация.
    cout << _x << endl;
}
```

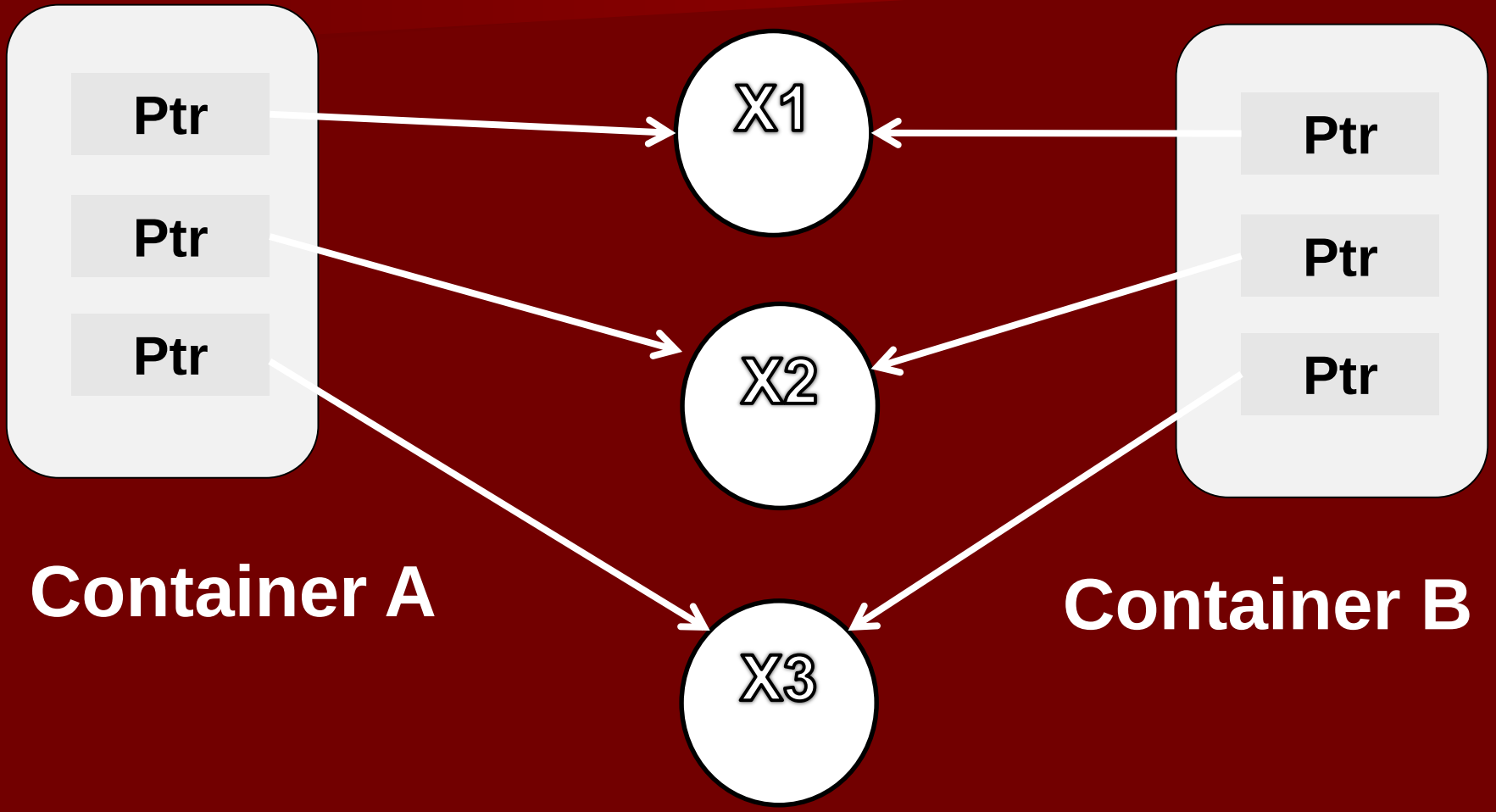
Интелигентни (smart) указатели (Visual Studio 2015+)

Дефиниция-обект, заграждащ указател и съдържащ функции за автоматично освобождаване на паметта.

Предназначение:

- Борба с утечките в динамичната памет;
- Сложни приложения с изключения;
- Паметта се освобождава в края на живота автоматично (чрез **деструктор**);
- Част от STL.

Интелигентни (smart) указатели (Visual Studio 2015+)



Интелигентни (smart) указатели (Visual Studio 2015+)

Същност-междинна версия между стеково-базирани и динамични
Особености:

- Създаване и инициализация-както указатели (референции) към стеково базирани променливи от тип указател;
- Не се изисква освобождаване на динамичната памет, заета от указателя (референцията) към променливата;
- Паметта се освобождава при излизане от обхват, както стеково-базирани променливи.

Интелигентни (smart) указатели (Visual Studio 2015+)

Важно правило:

- Да се създават на отделен ред, а не като формален параметър:
  - Цел – да се осигури правилото за излизане от обхват и да не се получи изтичане "боклук" в динамичната памет.

Използване:

- Да се използват за големи обекти и сложни начини за използване – предаване към други функции като референции.

Интелигентни (smart) указатели (Visual Studio 2015+)

Стъпки при използването:

- Декларация-като стеково базирана променлива-не се използва **new** или **malloc** за самия указател;
- Като тип се задава указател към типа на данните;
- Предаване се указател към създадената памет в конструктора на интелигентния указател;
- Използват се операторите за достъп до обекта: -> и *;
- **Няма нужда от освобождаване на паметта.**

Интелигентни (smart) указатели (Visual Studio 2015+)

Имат собствени функции, които се извикват с оператор «точка», напр. reset:

```
void SmartPointerDemo() {
   // Създаване на обект и предаването му на smart указател:
   std::unique_ptr<LargeObject> pLarge(new LargeObject());
   //Извикване на функция на обекта от клас LargeObject :
   pLarge->DoSomething();
   // Освобождаване на паметта предварително (преди блока):
   pLarge.reset();
   // Други оператори...
}
```

Интелигентни (smart) указатели (Visual Studio 2015+)

Видове, предоставени от STL:

- **unique_ptr;**
- **shared_ptr ;**
- **weak_ptr .**

Интелигентни (smart) указатели (Visual Studio 2015+)

unique_ptr : Особености и приложение:

- **Един собственик;**
- **Може да се предава на друга функция;**
- **Не се копира и споделя;**
- **Малък и ефективен-по големина и скорост-близък до обикновен указател;**
- **Поддържа референция към дясна част (стойност) – rvalue при вмъкване и добавяне в колекции.**

Интелигентни (smart) указатели (Visual Studio 2015+)

shared_ptr : Особености и приложение:

- Притежава брояч на референции;
- Използва се когато има присвояване на обикновен указател към множество собственици, напр. връщане на указателя при запазване на оригинала;
- Не се освобождава, докато не се излезе от обхвата на всички потребители;
- Големина-два указателя-към обекта и към паметта на референтния брояч.

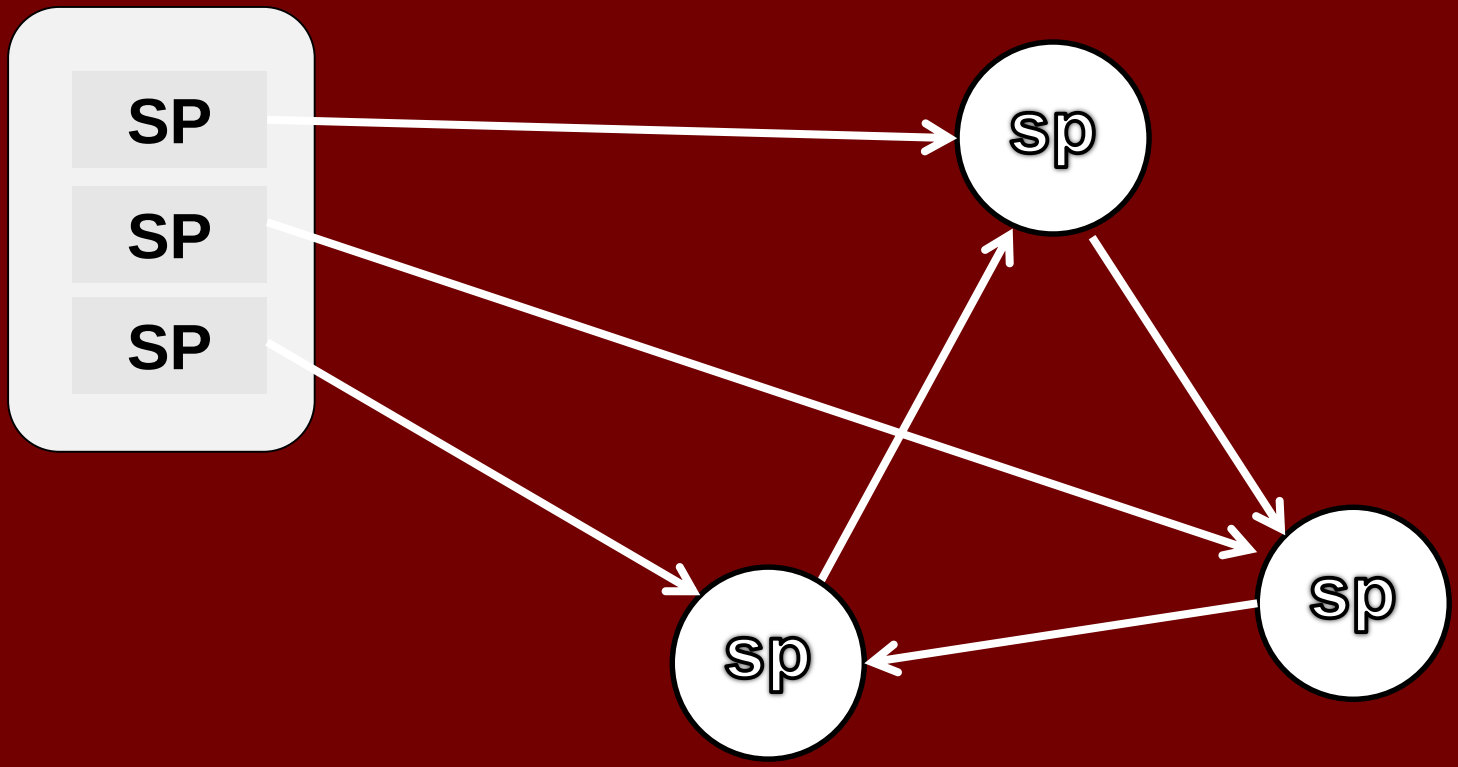
Интелигентни (smart) указатели (Visual Studio 2015+)

weak_ptr : Особености и приложение- специален (слабосвързан) shared_ptr :

- **Използва се когато има конюнкция от инстанции на един или повече shared_ptr;**
- **Няма брояч на референции на свързани помежду си shared_ptr;**
- **За достъп до свързани в цикъл инстанции;**
- **Решава проблеми за прекъсване на цикличните връзки между инстанциите от shared_ptr когато се освобождават.**

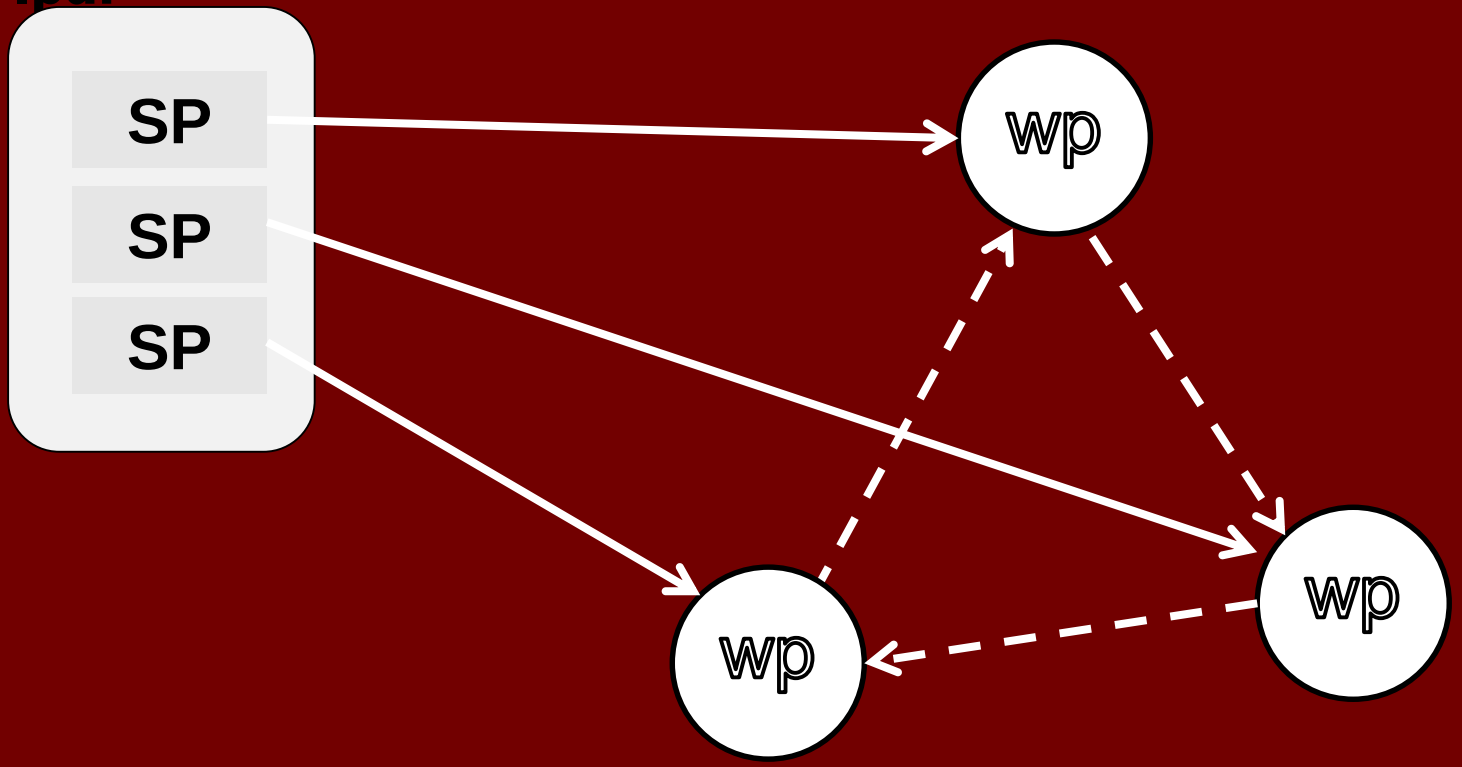
Интелигентни (smart) указатели (Visual Studio 2015+)

Циклични връзки с shared_ptr :



Интелигентни (smart) указатели (Visual Studio 2015+)

Разделяне на циклични връзки с weak_ptr : .pdf



За повече информация:
http://www.umich.edu/~eecs381/handouts/C++11_smart_ptrs.pdf

ВЪПРОСИ?